ROBOTİK DÜŞÜNME: KODDAN DAVRANIŞA

NECMİ GÜRSAKAL & ChatGPT

Bursa 2026

ROBOTİK DÜŞÜNME: KODDAN DAVRANIŞA

Necmi GÜRSAKAL & ChatGPT

Bursa2026

ARKA KAPAK YAZISI

Robot yapmak kolaylaştı.
Robotik düşünmek hâlâ zor.

Türkiye'de ve dünyada binlerce öğrenci sensör bağlıyor, motor sürüyor, çizgi izleyen araçlar yapıyor. Ama çok azı şu sorunun peşine düşüyor:
Bir makineye *davranış* nasıl kazandırılır?

Bu kitap, robotik eğitimine kökten farklı bir yerden başlıyor. Amaç, yalnızca çalışan sistemler kurmak değil; **öğrenen, uyum sağlayan ve karar verebilen sistemler tasarlamak**. Python ile başlayan yolculuk, PyTorch ile öğrenen modellere, oradan simülasyon dünyalarına ve en sonunda modern robotik platformlara uzanıyor. Bu süreçte okuyucu, robotu "kodlayan" değil, onu **eğiten** bir zihniyet kazanıyor.

Bu kitapta:

- Algoritmik düşünmeden davranış modellemeye geçişi
- Yapay zekâ ile robotiğin nasıl birleştiğini
- Simülasyondan gerçek dünyaya (sim-to-real) geçişin mantığını
- Modern robotik araçlarının nasıl kullanılacağını

adım adım öğreneceksiniz.

Bu bir proje kitabı değil.
Bu bir tarif kitabı hiç değil.

Bu kitap, bir çağ değişiminin eşiğinde, **makinelerle birlikte düşünmeyi öğrenmek isteyenler** için yazıldı.

Eğer bir robota ne yapacağını söylemekten fazlasını istiyorsanız, eğer bir makinenin *nasıl öğreneceğini* merak ediyorsanız, bu kitap sizinle konuşuyor.

ÖNSÖZ

Bir zamanlar makineler yalnızca kendilerine verilen komutları uygulardı. İnsan düşünür, makine yapardı. Bugün ise yeni bir dönemin eşiğindeyiz: artık makineler yalnızca çalışmıyor, öğreniyor; yalnızca hareket etmiyor, davranış geliştiriyor.

Robotik dünyası sessiz ama çok büyük bir dönüşüm geçiriyor. Klasik mühendislik yaklaşımı yerini giderek veriden öğrenen sistemlere bırakıyor. Kod satırlarıyla tek tek tarif edilen hareketlerin yerine, deneme-yanılma yoluyla davranış kazanan ajanlar geliyor. Humanoid robotlar, robot kolları, otonom sistemler ve dijital ikizler artık bilim kurgu değil; laboratuvarların ve şirketlerin günlük gerçeği hâline geliyor.

Ne var ki Türkiye'de bu dönüşümü bütünlüklü biçimde anlatan kaynak sayısı hâlâ oldukça sınırlı. Robotik çoğu zaman parçalar, kartlar ve küçük projeler düzeyinde ele alınıyor. Oysa modern robotik; yapay zekâ, simülasyon, kontrol teorisi, fizik motorları ve davranış öğrenmesini aynı çatı altında düşünmeyi gerektiriyor.

Türkiye'de robotik çoğu zaman hâlâ sensör bağlamak, motor döndürmek ya da yarışma projeleri üretmek olarak algılanıyor. Oysa dünya artık davranış öğrenen sistemlere geçiyor. NVIDIA Isaac Lab, reinforcement learning, simülasyon ortamları ve humanoid robotlar yeni dönemin merkezinde yer alıyor. Bu kitap, tam da bu kırılma noktasını Türkçe anlatmaya çalışıyor.

Elinizdeki kitap, okuyucuyu "kod yazan kişi" olmaktan "davranış tasarlayan kişi" olmaya taşımayı hedefliyor. Matematikten korkmadan, karmaşık teorilere boğulmadan ama düşünsel derinliği de kaybetmeden; robotik sistemlerin nasıl algıladığını, nasıl öğrendiğini ve neden hata yaptığını anlatmaya çalışıyor.

Kitap boyunca yalnızca robotları değil, yeni bir düşünme biçimini de keşfedeceğiz:

- Sensörlerin dünyayı nasıl temsil ettiğini,
- Geri beslemenin neden canlı sistemlere benzediğini,
- Reinforcement learning'in neden klasik programlamadan farklı olduğunu,
- Simülasyonda eğitilen bir robotun gerçek dünyaya nasıl aktarıldığını,
- Ve gelecekte insan ile makine arasındaki ilişkinin nasıl değişebileceğini…

Belki de bu kitabın en önemli amacı şudur: Robotik alanını yalnızca mühendislik meselesi olmaktan çıkarıp bir zihniyet dönüşümü olarak göstermek.

Çünkü önümüzdeki yıllarda robotik bilmek, yalnızca robot yapmak anlamına gelmeyecek. Tıpkı bilgisayar okuryazarlığı gibi, dünyayı anlayabilmenin temel araçlarından biri hâline gelecek.

Eğer bu kitap, bir öğrencinin geleceğe farklı bakmasını sağlar, bir öğretmenin yeni projeler üretmesine ilham verir ya da bir okurun “robotik aslında davranış tasarımıdır” düşüncesini keşfetmesine yardımcı olursa, amacına ulaşmış olacaktır.

Necmi Gürsakal & ChatGPT

10 Mayıs 2026

İÇİNDEKİLER

Önsöz
Giriş: Koddan Davranışa

BÖLÜM 1 — Robotik Nedir, Ne Değildir?

1.1 Bir Yanılgıyla Başlayalım
1.2 Robot = Makine + Davranış
1.3 Komutlarla Çalışan Dünya
1.4 Gerçek Dünya Neden Zordur?
1.5 Robotik Düşünmenin Doğuşu
1.6 Bir Örnek: Denge Kurmak
1.7 Robotik ≠ Arduino Projesi
1.8 Robotik ≠ Yapay Zekâ (Ama Ayrı da Değil)
1.9 Bu Kitapta Ne Yapacağız?
1.10 Bir Zihniyet Değişimi
1.11 Bölüm Özeti
1.12 Düşünme Soruları ve Cevapları
1.13 Mini Uygulama

BÖLÜM 2 — Öğrenen Sistemlere Giriş

2.1 Veri Nedir, Öğrenme Nedir?
2.2 Fonksiyon Öğrenmek: Sezgisel Bir Yaklaşım
2.3 Vektörler ve Matrislerle Düşünmek
2.4 PyTorch ile İlk Model
2.5 Basit Bir PyTorch Örneği
2.6 Hata, Kayıp ve İyileştirme: Bir Model Nasıl Öğrenir?
2.7 Ara Proje II: Görüntüden Anlam Çıkarmak (Basit Sınıflandırma)

BÖLÜM 3 — Robotik Sistemlerin Temelleri

3.1 Sensörler: Dünyayı Algılamak
3.2 Aktüatörler: Dünyaya Etki Etmek
3.3 Geri Besleme: Kontrolün Kalbi
3.4 Geri Besleme Nedir?
3.5 Açık Çevrim ve Kapalı Çevrim
3.6 Geri Besleme Neden “Kalp”tir?
3.7 Basit Kontrol Sistemi ve PID Mantığı
3.8 Fiziksel ve Sayısal Dünya Arasında
3.9 Ara Proje III: Çizgi İzleyen ve Engelden Kaçan Sistem

BÖLÜM 4 — Simülasyondan Gerçeğe

4.1 Neden Simülasyon?
4.2 Fizik Motorları ve Sanal Dünyalar
4.3 Ajan ve Ortam: Etkileşim Tasarımı
4.4 Öğrenme Yoluyla Kontrol (Reinforcement Learning'e Giriş)
4.5 Politika (Policy) Nedir? Davranış Nasıl Öğrenilir?
4.6 Ara Proje IV: Denge Kurmayı Öğrenen Ajan

BÖLÜM 5 — ISAAC LAB ve Öğrenen Robotlar

5.1 NVIDIA Isaac Lab ile Tanışma
5.2 Isaac Sim, PhysX ve Robotik Ekosistem
5.3 Simülasyonda Robot Eğitimi
5.4 Reinforcement Learning ile Davranış Öğretmek
5.5 Reward Function: Robotu Ne Motive Eder?
5.6 Paralel Öğrenme ve Hızlandırılmış Eğitim
5.7 Sim-to-Real: Simülasyondan Gerçek Dünyaya Geçiş
5.8 Domain Randomization ve Dayanıklı Davranış
5.9 Humanoid Robotlar Nasıl Eğitiliyor?
5.10 Robot Kolları, Hareket ve Manipülasyon
5.11 Isaac Lab ile Geleceğin Robotları
5.12 Ara Proje V: Nesne Kavrayan Robot

BÖLÜM 6 — Modern Robotik ve Davranış

6.1 Karmaşık Davranışların Öğrenilmesi
6.2 Hata, Belirsizlik ve Dayanıklılık
6.3 Otonomi ve Karar Verme
6.4 Çoklu Ajan Sistemleri
6.5 Yapay Zekâ ve Robotik Geleceği
6.6 İnsan–Makine İşbirliği
6.7 Robotik Düşünmenin Felsefesi

EKLER

EK A — Matematiksel Sezgiler (Vektör, Trigonometri, Türev)
EK B — GitHub ile Proje Yönetimi
EK C — Öğretmenler İçin Kılavuz
EK D — Kulüpler İçin 1 Yıllık Uygulama Planı

KAYNAKÇA

GİRİŞ: KODDAN DAVRANIŞA

Uzun süre makinelerle kurduğumuz ilişki basitti: Biz söylerdik, onlar yapardı.

Bir robotu programlamak, ona adım adım ne yapacağını anlatmaktı. “İleri git”, “sağa dön”, “engel varsa dur”... Bu komutlar dünyası, robotik eğitiminin de temelini oluşturdu. Öğrenciler sensörleri bağladı, motorları kontrol etti, kod yazdı ve çalışan sistemler üretti.

Ama bir noktada bu yaklaşım sınırına ulaştı.

Çünkü gerçek dünya, önceden yazılmış komut listelerine sığmayacak kadar karmaşıktı. Bir robot, her olası durumu önceden bilemez. Her senaryo için ayrı bir kod yazmak mümkün değildir. İşte bu noktada, robotik yeni bir yöne evrildi:

Makinelere ne yapacaklarını söylemek yerine, onlara **nasıl öğreneceklerini öğretmeye** başladık.

Bu değişim, yalnızca teknik bir gelişme değil, aynı zamanda zihinsel bir kırılmadır. Artık mesele kod yazmak değil; **davranış üretmektir**. Bir robotun denge kurmayı, yürümeyi ya da bir nesneyi kavramayı “öğrenmesi”, onu klasik anlamda programlamaktan tamamen farklı bir süreçtir.

Bu kitap, tam olarak bu dönüşümün içinden yazıldı.

Burada Python öğreneceksiniz, evet. Ama Python bir amaç değil, bir araç olacak. PyTorch ile modeller kuracaksınız, ama asıl hedef model değil, **öğrenme fikrini kavramak** olacak. Simülasyon ortamlarında robotlar eğiteceksiniz, ama asıl kazanımınız bir robotu kontrol etmek değil, **ona davranış kazandırmak** olacak.

Çünkü modern robotik, üç büyük alanın kesişiminde doğar:

- Hesaplama (programlama)
- Öğrenme (yapay zekâ)
- Fiziksel dünya (robotik sistemler)

Bu kitap, bu üç alanı lise düzeyinden başlayarak erişilebilir ama yüzeysel olmayan bir şekilde bir araya getirir.

Belki de en önemlisi şudur:

Bu kitap size hazır çözümler vermez. Size doğru soruları sorar.

Bir robot neden dengesini kaybeder?
Bir sistem nasıl geri bildirimle kendini düzeltir?
Bir makine, hiç görmediği bir duruma nasıl tepki verir?

Bu soruların peşinden gitmek, robotik öğrenmenin en güçlü yoludur.

Eğer bu kitabın sonunda hâlâ sadece “çalışan” robotlar yapıyorsanız, bir şeyleri kaçırmışsınız demektir.
Ama eğer makinelerin nasıl öğrendiğini, nasıl uyum sağladığını ve nasıl karar verdiğini anlamaya başladıysanız, doğru yoldasınız.

Çünkü gelecek, komut verenlerin değil, **davranış tasarlayanların** olacak.

BÖLÜM 1 — ROBOTİK NEDİR, NE DEĞİLDİR?

Bir robot gördüğümüzde genellikle onu hareket eden bir makine olarak tanımlarız. Tekerlekleri dönen bir araç, kolunu oynatan bir mekanizma ya da sensörlerle çevresini algılayan bir sistem… Bunların hepsi robot gibi görünür. Ama robotik, görünenin ötesinde bir kavramdır. Bu bölümün temel sorusu şudur: **Bir sistemi gerçekten robot yapan nedir?**

Robotik, basitçe donanım parçalarının bir araya gelmesi değildir. Bir robotu robot yapan şey, üç temel bileşenin birlikte çalışmasıdır: algı, karar ve eylem. Bir robot, sensörleri aracılığıyla dünyayı algılar; bu bilgiyi kullanarak bir karar verir ve ardından fiziksel bir eylem gerçekleştirir. Bu üçlü yapı olmadan bir sistem yalnızca mekanik bir düzenektir.

Lise düzeyinde yapılan birçok çalışmada robotik, çoğu zaman elektronik devre kurmak ve belirli komutları yazmak olarak anlaşılır. Örneğin bir sensör belirli bir mesafenin altını ölçtüğünde motoru durdurmak gibi basit kurallar tanımlanır. Bu tür sistemler çalışır, hatta etkileyici görünebilir. Ancak burada önemli bir sınırlılık vardır: Bu sistemler yalnızca önceden tanımlanmış durumlara tepki verir. Yani “düşünmezler”, sadece “uygularlar”.

Gerçek dünya ise bu kadar basit değildir. Sensörler hatalı ölçüm yapabilir, ortam değişken olabilir ve her durum önceden tahmin edilemez. Bu nedenle, her olasılığı kapsayan bir komut listesi yazmak pratikte mümkün değildir. İşte bu noktada robotik anlayışı değişir. Artık mesele, robota ne yapacağını söylemek değil, onun nasıl öğreneceğini tasarlamaktır.

Modern robotik bu dönüşüm üzerine kuruludur. Bir robotun denge kurmayı, yürümeyi ya da bir nesneyi kavramayı öğrenmesi, klasik programlamadan farklı bir yaklaşım gerektirir. Bu yaklaşımda robot, deneme-yanılma yoluyla deneyim kazanır ve zamanla daha iyi davranışlar geliştirir. Böylece robot, yalnızca verilen komutları uygulayan bir sistem olmaktan çıkar; çevresine uyum sağlayabilen bir varlığa dönüşür.

Bu noktada önemli bir ayrım ortaya çıkar: Robotik, sadece donanım değildir; yapay zekâ da sadece yazılım değildir. Modern robotik, bu iki alanın kesişiminde yer alır. Fiziksel dünya ile hesaplama gücünün birleştiği bu alan, geleceğin en kritik disiplinlerinden biridir.

> Bu kitapta robotiği bir “proje yapma” etkinliği olarak değil, bir **düşünme biçimi** olarak ele alacağız. Amaç, çalışan sistemler kurmak kadar, bu sistemlerin nasıl davrandığını anlamaktır. Çünkü asıl mesele, bir robotu hareket ettirmek değil; ona anlamlı ve esnek davranışlar kazandırmaktır.

Bu bölümün sonunda şu farkı net olarak görmelisiniz: Her robot bir makinedir, ama her makine bir robot değildir. Aradaki fark ise tek bir kelimede saklıdır: **davranış**.

1.1 Bir Yanılgıyla Başlayalım

Bir lise robotik kulübüne girdiğinizde genellikle aynı manzarayla karşılaşırsınız: masaların üzerinde dağılmış kartlar, kablolar, sensörler, motorlar ve küçük tekerlekli araçlar. Bir grup öğrenci bir şeyleri lehimlemeye çalışır, bir başkası kod ekranına bakar, bir diğeri elindeki parçanın nereye takılacağını anlamaya çalışır. Ortam canlıdır, üretkendir ve dışarıdan bakıldığında oldukça etkileyicidir.

Bir süre sonra biri yüksek bir sesle söyler: “Hocam, robot yaptık.”

Bu cümle genellikle bir başarı ilanı gibi duyulur. Gerçekten de öyledir; çünkü bir şey çalıştırılmıştır, bir sistem hayata geçirilmiştir. Ama tam burada durup sormak gerekir: **Gerçekten bir robot mu yaptınız?**

Bu sorunun basit bir cevabı yoktur. Çünkü “robot yapmak” ifadesi, çoğu zaman teknik bir eylemden çok bir algıyı temsil eder. Çalışan bir düzenek görmek, onun robot olduğu fikrini otomatik olarak doğurur. Oysa robotik, sadece çalışan şeyler üretmek değildir.

Birçok lise projesinde süreç oldukça benzer ilerler. Önce bir sensör seçilir. Örneğin mesafe ölçen bir sensör. Ardından bir motor bağlanır. Daha sonra küçük bir kontrol kartı devreye eklenir. Son adımda ise birkaç koşul yazılır:

- Eğer mesafe şu değerin altına düşerse dur
- Değilse ilerle

Sistem çalıştırıldığında gerçekten de bir davranış ortaya çıkar. Araç ilerler, engel görünce durur, bazen yön değiştirir. Dışarıdan bakıldığında bu oldukça “akıllı” bir davranış gibi görünebilir. Ama burada kritik bir ayrım vardır: Bu sistem gerçekten “anlıyor” mu, yoksa sadece “tepki mi veriyor”?

Bu soruyu netleştirmek önemlidir. Çünkü bu tür sistemler aslında dünyayı anlamaz. Onlar yalnızca önceden tanımlanmış koşullara verilen otomatik tepkileri uygular. Yani bir anlamda, çok gelişmiş bir “eğer–ise” makinesidir.

Bu noktada robotik ile otomasyon arasındaki fark bulanıklaşır. Bir sistemin çalışıyor olması, onun robot olduğu anlamına gelmez. Çalışan bir çamaşır makinesi de belirli koşullara göre

hareket eder; ama kimse ona robot demez. Aynı şekilde, belirli sensör değerlerine göre hareket eden bir araç da her zaman robotik bir sistem değildir.

> Gerçek robotik sistemler, yalnızca önceden yazılmış kurallara bağlı değildir. Onlar, değişen koşullara uyum sağlayabilen, belirsizlikle baş edebilen ve zaman içinde davranışını geliştirebilen sistemlerdir. İşte fark tam burada ortaya çıkar.

Bir robotun "davranış üretmesi" ile "komut uygulaması" arasında büyük bir uçurum vardır. Komut uygulayan bir sistem, yalnızca yazılanı yapar. Davranış üreten bir sistem ise çevresiyle etkileşime girer, geri bildirim alır ve buna göre kendini yeniden düzenler.

Bu yüzden robotik kulüplerinde sıkça görülen bir yanılgı vardır: Çalışan bir düzenek, otomatik olarak "öğrenen" ya da "akıllı" bir sistem sanılır. Oysa çoğu zaman bu sistemler sabittir; değişmezler, gelişmezler, sadece tetiklendiklerinde belirli bir çıktıyı üretirler.

İşte bu kitabın temel iddiası burada başlar. Robotik, sadece "bir şeyin çalışması" değildir. Robotik, bir sistemin **çevresiyle kurduğu ilişkiyi tasarlama sanatıdır**. Bu ilişki, basit kurallarla da kurulabilir, öğrenen modellerle de. Ama sonuç aynı değildir.

Dolayısıyla şu soruya geri dönmek gerekir: Bir robot yaptığınızda, gerçekten ne yapmış oluyorsunuz?

Bir motoru döndürmek mi?
Bir sensörü okumak mı?
Bir kartı programlamak mı?

Yoksa daha fazlası mı?

Belki de asıl mesele şudur:
Bir sistemin hareket etmesini sağlamak kolaydır. Ama o sistemin **nasıl davranacağını tasarlamak**, işin gerçek robotik kısmıdır.

Bu bölüm boyunca sık sık şu ayrımı hatırlayacağız:
Çalışan sistemler vardır.
Ve davranış üreten sistemler vardır.

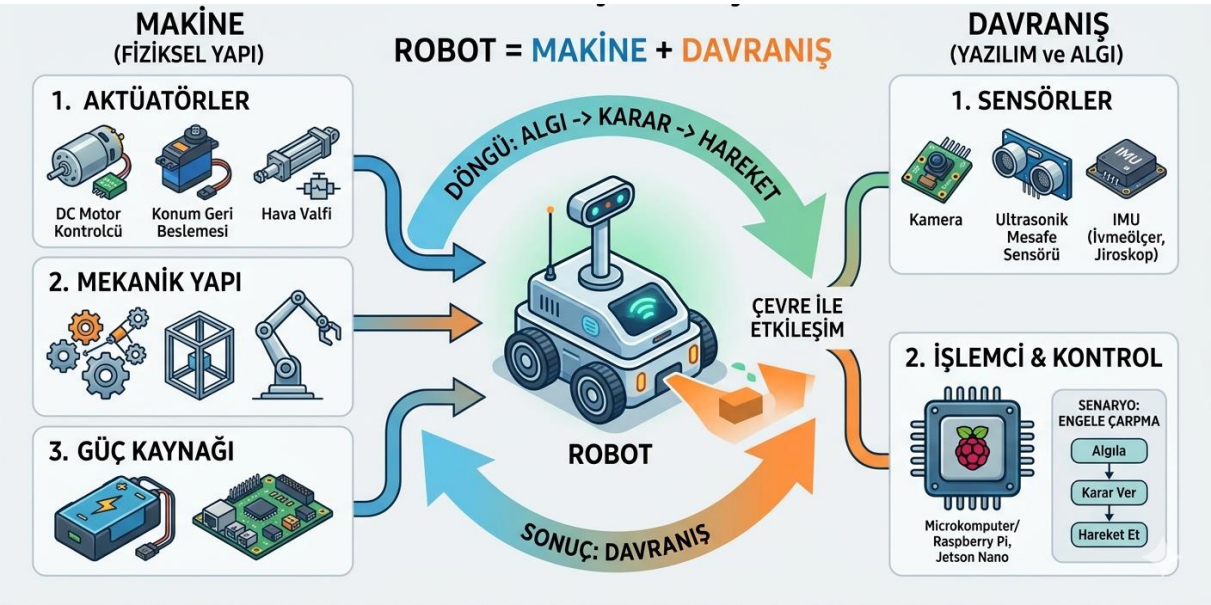
Aralarındaki fark, robotik düşünmenin başladığı yerdir.

1.2 Robot = Makine + Davranış

Bir robotu anlamanın en basit ama aynı zamanda en derin yolu, onu parçalarına ayırarak düşünmektir. Çünkü yüzeyde karmaşık ve bütünleşik görünen her sistem, aslında daha küçük ve anlaşılır bileşenlerin bir araya gelmesiyle oluşur. Robotik de bu açıdan bir istisna değildir; hatta belki de bu parçalı yapının en görünür olduğu alanlardan biridir. Eğer robot kavramını

gerçekten anlamak istiyorsak, onu tek bir “cihaz” olarak değil, sürekli çalışan bir “yapı” olarak düşünmek gerekir.

Şekil 1. Robotik Bileşenler Şeması



Şekil 1’de Sol tarafta robotun fiziksel yapısı yer alıyor: aktüatörler hareketi sağlar, mekanik yapı gövdeyi oluşturur ve güç kaynağı sistemi çalıştırır. Sağ tarafta ise robotun “davranış” tarafı bulunur: sensörler çevreyi algılar, işlemci verileri yorumlar ve karar mekanizması uygun hareketi seçer. Ortadaki döngü, modern robotiğin temelini özetler: algıla → karar ver → hareket et. Böylece robot çevresiyle etkileşime girer ve ortaya davranış çıkar. Bu yapıyı en yalın biçimde ifade eden çerçeve şudur:

Robot = Algı + Karar + Eylem

Bu eşitlik yalnızca bir tanım değildir; aynı zamanda bir düşünme modelidir. Bir sistemin robot sayılabilmesi için bu üç bileşenin birlikte, süreklilik içinde ve döngüsel olarak çalışması gerekir. Bunlardan biri eksikse ortaya çıkan şey robot değil, yalnızca bir makine, bir araç ya da bir otomasyon düzeneğidir.

Robotik düşünmenin başlangıç noktası tam olarak burasıdır: Bir sistemi parçalarına ayırarak değil, bu parçaların nasıl bir döngü oluşturduğunu anlayarak kavramak.

Algı (Perception): Dünyayı Anlamak

Robotik sistemin ilk bileşeni algıdır. Algı, bir robotun içinde bulunduğu dünyayı “hissetmesini” sağlayan mekanizmadır. Ancak burada “hissetmek” kelimesi insanî bir benzetmedir; robotlar dünyayı hissetmez, ölçer.

İnsan için göz, kulak ve dokunma duyusu ne ifade ediyorsa, robotlar için sensörler aynı işlevi görür. Kamera görüntü üretir, LiDAR, “Light Detection and Ranging” ifadesinin kısaltmasıdır.

Türkçe karşılığıyla “Işık ile Algılama ve Uzaklık Ölçümü” gibi düşünebilirsin. LIDAR mesafe ölçer, sıcaklık sensörü çevresel değişimi sayısal hale getirir, ivmeölçer hareketi kaydeder. Ancak tüm bu veriler ham ve anlamsızdır; robot için dünya, doğrudan anlamlı nesnelerden değil, sayılardan oluşur.

Bu nedenle algı, fiziksel dünyanın sayısal bir temsile dönüştürülmesidir. *Bir robot önündeki nesneyi “görmez”, onun uzaklığını bir sayı olarak alır. O nesnenin “engel” olup olmadığını ise doğrudan bilemez; bunu ancak veriden çıkarım yaparak anlayabilir.*

Ancak algı süreci yalnızca veri toplamak değildir. Aynı zamanda belirsizlikle yaşama sanatıdır. Çünkü sensörler kusursuz değildir. Gürültü vardır, ölçüm hataları vardır, çevre sürekli değişir. Aynı nesne farklı zamanlarda farklı ölçülebilir. Bu nedenle algı, kesinlik değil, olasılık üretir. Robotik sistemler bu yüzden dünyayı “gerçek haliyle” değil, “tahmini haliyle” temsil eder. Bu temsil, sistemin tüm karar mekanizmasının temelini oluşturur.

Karar (Decision): Ne Yapılacağını Belirlemek

Algıdan sonra gelen ikinci bileşen karardır. Bir robot çevresini algıladıktan sonra bu bilgiyi nasıl kullanacağını belirlemek zorundadır. İşte bu noktada “davranışın zihinsel kısmı” devreye girer. Karar verme süreci, algılanan verinin bir eyleme dönüştürülmesidir. Ancak bu süreç göründüğünden çok daha karmaşıktır. Çünkü aynı durum için birden fazla doğru eylem olabilir. Robot, bu seçenekler arasından birini seçmek zorundadır.

Basit sistemlerde bu seçim genellikle kurallarla yapılır. Örneğin engel varsa dur, yol açıksa ilerle gibi ifadeler kullanılır. Bu tür sistemler deterministiktir; yani aynı girdiye her zaman aynı çıktıyı üretirler. Bu yaklaşım küçük ve kontrollü ortamlarda işe yarar.

Ancak gerçek dünya bu kadar basit değildir. Sensör verileri çelişkili olabilir, ortam dinamik olabilir, hatta “doğru karar” bile bağlama göre değişebilir. Bu noktada klasik kural tabanlı sistemler yetersiz kalır.

Modern robotikte karar verme artık yalnızca önceden yazılmış kuralların uygulanması değildir. Öğrenme tabanlı sistemler devreye girer. Robot, geçmiş deneyimlerinden öğrenerek hangi durumda hangi davranışın daha iyi sonuç verdiğini keşfeder. Böylece karar verme süreci statik olmaktan çıkar, dinamik bir yapıya dönüşür.

Karar, artık yalnızca bir hesaplama değil, bir öğrenme sonucudur.

Eylem (Action): Dünyaya Müdahale Etmek

Üçüncü bileşen eylemdir. Bu, robotun fiziksel dünyayla doğrudan temas kurduğu noktadır. Motorların çalışması, tekerleklerin hareket etmesi, robot kolunun bir nesneyi kavraması gibi tüm fiziksel çıktılar bu kategoriye girer.

Eylem, robotun iç dünyasından dış dünyaya açılan kapıdır. Algı ve karar ne kadar gelişmiş olursa olsun, eğer eylem yoksa sistem tamamlanmamış olur. Böyle bir sistem yalnızca düşünür ama hiçbir şeyi değiştiremez.

Ancak eylem yalnızca hareket üretmek değildir. Eylem, aynı zamanda dünyayı değiştirmektir. Bir robot hareket ettiğinde çevresiyle olan ilişkisi değişir. Yeni bir konuma geçer, yeni veriler toplar ve bu veriler yeni kararları tetikler.

Bu nedenle robotik sistemler kapalı değil, sürekli döngü halinde çalışan yapılardır:

Algı → Karar → Eylem → Yeni Algı

Bu döngü, robotun durağan bir makine değil, sürekli etkileşim halinde olan bir sistem gibi davranmasını sağlar. Robotun "yaşıyor gibi" görünmesinin nedeni de bu sürekli geri besleme mekanizmasıdır.

Üçlü Döngünün Önemi

Robotik sistemlerin gerçek gücü, onu oluşturan parçaların tek tek varlığında değil, bu parçaların sürekli ve kesintisiz bir döngü içinde birlikte çalışmasında ortaya çıkar. Algı, karar ve eylem ayrı ayrı ele alındığında her biri kendi içinde anlamlıdır; ancak robotik davranış dediğimiz şey, ancak bu üç bileşen birbiriyle etkileşime girdiğinde ortaya çıkar. Bu nedenle robotik sistemleri anlamak, aslında bir döngüyü anlamaktır.

Eğer bu üç bileşen arasında bir kopukluk varsa, sistemin robotik niteliği ortadan kalkar. Çünkü robot dediğimiz şey sabit bir yapı değil, sürekli kendini güncelleyen bir etkileşim mekanizmasıdır. Bu etkileşim kesildiğinde geriye yalnızca parçalar kalır, ancak davranış ortadan kalkar.

Sadece algıdan oluşan bir sistem düşünelim. Böyle bir sistem çevresini ölçebilir, veri toplayabilir, sensörlerden gelen bilgileri kaydedebilir. Ancak bu sistem hiçbir şey yapmaz. Eyleme dönüşmeyen algı, yalnızca bir gözlem mekanizmasıdır. Bir kamera sistemi ya da bir ölçüm cihazı gibi çalışır. Dünya hakkında bilgi üretir ama bu bilgiyi kullanmaz. Dolayısıyla böyle bir yapı robot değil, pasif bir veri toplayıcıdır.

Benzer şekilde sadece eylem üreten bir sistem de robot değildir. Eğer bir sistem çevresini anlamadan hareket ediyorsa, bu kör bir mekanizmadır. Örneğin sürekli ileri giden ama önünde ne olduğunu bilmeyen bir motor sistemi, rastgele bir fiziksel süreçten ibarettir. Eylem vardır ama yönlendiren bir zihin yoktur. Bu nedenle davranış üretmez, yalnızca hareket üretir.

Sadece karar mekanizmasına sahip bir sistem ise daha soyut bir yapıdır. Bu tür bir sistem hesaplama yapabilir, analiz üretebilir, hatta optimal çözümler bulabilir. Ancak bu kararlar ne algıya dayanır ne de eyleme dönüşür. Dolayısıyla sistem kapalı bir düşünce yapısı içinde kalır. Ne dünyayı etkiler ne de dünyadan etkilenir. Bu nedenle robotik anlamda işlevsizdir.

Bu üç eksiklik bize önemli bir gerçeği gösterir: robotik sistemin değeri, bileşenlerinde değil, bu bileşenlerin döngüsel olarak birbirine bağlanmasındadır. Algı bilgi üretir, karar bu bilgiyi yorumlar, eylem ise dünyayı değiştirir. Bu değişim yeniden algıyı tetikler ve döngü tekrar başlar. İşte bu sürekli geri besleme, robotik sistemin "yaşayan" bir yapıya dönüşmesini sağlar.

Bu noktada kritik bir ayrım ortaya çıkar: bir sistem yalnızca çalışıyor mu, yoksa davranış mı üretiyor? Çalışmak ile davranış üretmek aynı şey değildir. Çalışan bir sistem pasif olabilir; belirli bir işlevi yerine getirir ama çevresiyle etkileşim kurmaz. Oysa davranış üreten sistem, çevreyi değiştirir ve aynı zamanda çevreden etkilenir.

Davranış, robotik sistemlerin en temel özelliğidir. Eğer bir sistem davranış üretmiyorsa, ne kadar karmaşık olursa olsun robot değildir. Sadece mekanik bir düzeneğin daha gelişmiş bir versiyonudur.

Bu nedenle robotik düşünmede temel soru şudur: sistem ne yapıyor değil, sistem nasıl bir etkileşim içinde yaşıyor? Çünkü robotik, yalnızca işlem yapmak değil, dünyayla sürekli ilişki kurmaktır.

Robotun Yeniden Tanımı

Bu çerçeveyle birlikte robotu yeniden tanımlamak artık kaçınılmaz hale gelir. Geleneksel anlayışta robot, belirli görevleri yerine getiren otomatik bir makine olarak görülüyordu. Fabrika bantlarında aynı hareketi tekrar eden mekanik kollar, önceden programlanmış komutları uygulayan cihazlar ya da belirli kurallarla çalışan otomasyon sistemleri uzun süre robot kavramının merkezinde yer aldı. Ancak günümüz teknolojik gelişmeleri bu tanımı önemli ölçüde değiştirmiştir.

Bugün bir robotu yalnızca “hareket eden makine” olarak tanımlamak yetersizdir. Modern robot, çevresini algılayan, bu algıları veriye dönüştüren, verileri yorumlayan, karar veren ve bu kararları fiziksel eylemlere dönüştüren bütünleşik bir sistemdir. Başka bir ifadeyle robot, sensörler, hesaplama mekanizmaları, öğrenme süreçleri ve fiziksel hareket kabiliyetinin birleştiği dinamik bir yapıdır.

Bu nedenle robotun özü artık yalnızca mekanikte değil, bilgi işlem ve karar verme süreçlerinde yatmaktadır. Kameralar, LIDAR sistemleri, mikrofonlar, dokunma sensörleri ve diğer algılama araçları robotun “duyuları” haline gelirken; yapay zekâ modelleri, olasılıksal hesaplamalar ve öğrenme algoritmaları onun “yorumlama” mekanizmasını oluşturmaktadır. Motorlar, aktüatörler ve hareket sistemleri ise bu zihinsel sürecin fiziksel dünyadaki karşılığıdır.

Ancak bu tanımın altında daha derin bir gerçek vardır: Robot, aslında davranış üreten bir sistemdir. Onu özel kılan şey yalnızca hareket etmesi değil, çevresel koşullara tepki verebilmesidir. Bir robotun davranışı tamamen önceden belirlenmiş kurallardan doğabilir; örneğin belirli bir engeli gördüğünde yön değiştirebilir. Fakat daha gelişmiş sistemlerde davranış, öğrenme süreçleri sonucunda ortaya çıkar. Robot deneyimlerinden öğrenebilir, hata yapabilir, performansını geliştirebilir ve yeni durumlara uyum sağlayabilir.

Bu dönüşüm, robot kavramını klasik otomasyondan ayıran temel noktadır. Geleneksel makineler yalnızca programlandıkları işi yaparken, modern robotik sistemler belirsizlik içinde çalışabilir. Çünkü gerçek dünya kusursuz değildir. Işık değişir, nesneler yer değiştirir, insanlar

tahmin edilemez davranır ve çevre sürekli dönüşür. Bu nedenle çağdaş robotlar yalnızca komut uygulayan sistemler değil; aynı zamanda belirsizlikle başa çıkan uyarlanabilir varlıklardır.

Burada önemli olan bir başka nokta da robotun artık yalnızca fiziksel bir cihaz olarak düşünülmemesidir. Robotik düşünce, davranış üreten her türlü akıllı sistemi kapsayacak biçimde genişlemektedir. Bir otonom araç, depo robotu, cerrahi robot, insansı robot ya da endüstriyel kol farklı görünümlere sahip olabilir; ancak hepsi aynı temel mantığı paylaşır: algıla, yorumla, karar ver ve hareket et. Bu dört aşama modern robotun bilişsel döngüsünü oluşturur.

Dolayısıyla robotu yeni çağda şu şekilde tanımlamak mümkündür:

> Bir robot, çevresini sürekli algılayan, bu verileri yorumlayan, amaçları doğrultusunda karar veren ve fiziksel ya da dijital eylemler üreten uyarlanabilir bir davranış sistemidir.

Bu tanım robotu yalnızca mekanik bir araç olmaktan çıkarır ve onu çevresiyle sürekli ilişki kuran dinamik bir aktör haline getirir. Geleceğin robotları yalnızca fabrikalarda çalışan makineler olmayacaktır. Evlerde yardımcı olan sistemler, hastanelerde karar destek sağlayan cihazlar, tarım alanlarında çalışan otonom araçlar, hatta dijital dünyada görev yapan yazılım ajanları bile bu geniş robotik ekosistemin parçası haline gelecektir.

Sonuçta robotik, artık yalnızca “hareket eden makineler” bilimi değildir. Robotik; algı, öğrenme, karar verme ve davranış üretme süreçlerini bir araya getiren yeni bir düşünme biçimidir. Bu nedenle geleceğin en önemli sorularından biri yalnızca “Robot ne yapabilir?” değil, aynı zamanda “Robot nasıl öğrenir, nasıl karar verir ve insanlarla nasıl birlikte yaşar?” sorusu olacaktır.

1.3 Komutlarla Çalışan Dünya

Klasik robotik, uzun yıllar boyunca oldukça net ve anlaşılır bir mantıkla ilerledi. Bu yaklaşımın temelinde basit ama güçlü bir fikir vardır: Bir sistemi davranışa zorlamak için ona açık ve kesin kurallar vermek.

Bu dünya şu cümleyle özetlenebilir:

“Eğer sensör şunu görürse, bunu yap.”

Bu ifade, robotik düşüncenin en temel yapı taşlarından biridir. Çünkü burada belirsizlik yoktur; her şey koşullara bağlanmıştır. Sistem, dünyayı anlamaya çalışmaz, sadece ona verilen kuralları uygular.

Bunu en basit haliyle bir Python dilinde yazılmış bir kod parçasında görebiliriz:

```
if distance < 10:
    stop()
```

else:

move_forward()

Bu sistemde distance değişkeni, robotun önündeki bir nesneye olan mesafeyi temsil eder. Bu değer genellikle bir sensörden (örneğin ultrasonik sensör veya LiDAR) gelir.

Kod şu mantıkla çalışır:

- Eğer robotun önündeki mesafe **10 birimden küçükse**, yani bir nesne çok yakınsa, robot **durur** (stop()).
- Eğer mesafe **10 birimden büyükse**, yani yol açıksa, robot **ileri hareket eder** (move_forward()).

Bu birkaç satırlık yapı, aslında klasik robotik felsefesinin tamamını temsil eder. Bir sensör mesafeyi ölçer, bir koşul değerlendirilir ve sonuç olarak bir eylem gerçekleştirilir. Sistem çalışır, davranış üretir ve dışarıdan bakıldığında "akıllı" gibi görünebilir.

Ama burada kritik bir soru ortaya çıkar: Bu sistem gerçekten "anlıyor" mu, yoksa sadece tepki mi veriyor? Bu soru, robotik düşünmenin en önemli ayrım noktalarından biridir. Çünkü "anlamak" ile "tepki vermek" arasında çok büyük bir fark vardır.

Tepki Veren Sistemler

Tepki veren sistemler, robotik ve otomasyon dünyasının en temel ama aynı zamanda en sınırlı yapı taşlarından biridir. Bu tür sistemler dış dünyayı anlamaya çalışmaz, onu modellemez ve öğrenmez. Onların çalışma prensibi oldukça doğrudandır: belirli bir durum oluştuğunda, o duruma karşılık gelen önceden tanımlanmış bir eylem tetiklenir. Bu nedenle tepki veren sistemler, çoğu zaman "karar veren" bir yapıdan ziyade "durum–eylem eşleştirme mekanizması" olarak düşünülebilir.

Bu sistemlerde davranış, mühendis tarafından önceden yazılmış kuralların sonucudur. Sistem, karşılaştığı durumu analiz etmez; yalnızca o durumun daha önce tanımlanmış olup olmadığına bakar. Eğer durum tanımlıysa ilgili eylem çalıştırılır, tanımlı değilse sistem ya hiçbir şey yapmaz ya da varsayılan bir davranışa geçer. Bu yapı, özellikle basit kontrol problemlerinde oldukça etkilidir ve mühendislik açısından kolay uygulanabilir bir çözüm sunar.

Ancak bu kolaylık, önemli bir sınırlılığı da beraberinde getirir: öğrenme yoktur. Tepki veren sistemler deneyimden beslenmez, dolayısıyla zaman içinde gelişmezler. Bugün nasıl davranıyorlarsa, yarın da aynı şekilde davranırlar. Bu durum onları sabit ve öngörülebilir kılar, ancak aynı zamanda adaptasyon yeteneklerini tamamen ortadan kaldırır.

Bir diğer kritik nokta ise genelleme eksikliğidir. Bu sistemler yalnızca açıkça tanımlanmış durumlar için çalışır. Eğer sistem daha önce tanımlanmamış bir durumla karşılaşırsa, onu yorumlayamaz. Bu nedenle dünya, bu sistemler için sonsuz sayıda "eğer–ise" kuralından oluşan

bir listeye indirgenir. Her yeni durum için yeni bir kural eklenmesi gerekir. Bu da sistemin ölçeklenmesini zorlaştırır ve zamanla yönetilemez hale getirebilir.

Esneklik Eksikliği

Komut tabanlı veya tepki veren sistemlerin en belirgin problemi esneklik eksikliğidir. Esneklik, bir sistemin değişen koşullara uyum sağlayabilme kapasitesidir ve bu tür sistemlerde bu kapasite neredeyse yoktur. Çünkü davranış, çevreden değil tasarımcıdan gelir.

Bu sistemler genellikle belirli bir ortam için tasarlanır ve o ortamın varsayımlarına göre optimize edilir. Örneğin düz bir zeminde hareket eden bir robot düşünelim. Bu robotun tüm kontrol mantığı, zeminin sabit olduğu varsayımına dayanır. Ancak zemin eğimli hale gelirse, sürtünme değişirse veya sensör verileri bozulursa sistem beklenmedik hatalar üretmeye başlar.

Bunun nedeni oldukça basittir: sistem dünyayı öğrenmez, yalnızca tanımlar. Dünya hakkında sahip olduğu bilgi, geliştirici tarafından yazılmış sınırlı kurallardan ibarettir. Bu kuralların dışına çıkıldığında sistemin davranışı bozulur.

Gerçek dünya ise hiçbir zaman sabit değildir. Sürekli değişir, belirsizlik içerir ve çoğu zaman öngörülemezdir. Bu nedenle sabit kurallara dayalı sistemler, bu değişkenlik karşısında kırılgan hale gelir. Küçük bir çevresel değişiklik bile sistemin davranışını tamamen yanlış yöne sürükleyebilir.

Yeni Durumlar Problemi

Komut tabanlı sistemlerin en ciddi sorunlarından biri “yeni durumlar” problemidir. Bir sistem tasarlanırken geliştirici, belirli senaryoları öngörür ve bu senaryolar için uygun kurallar yazar. Ancak gerçek dünya, sonsuz çeşitlilikte durum üretir ve bu durumların tamamını önceden tahmin etmek mümkün değildir.

Örneğin bir robotun sensörü beklenmedik bir değer üretebilir. Ortam ışığı aniden değişebilir, bir engel alışılmadık bir açıyla yerleştirilmiş olabilir ya da sistem daha önce hiç karşılaşmadığı bir kombinasyonla yüzleşebilir. Bu tür durumlarda sistemin vereceği tepki ya yanlış olur ya da hiç oluşmaz.

Bunun nedeni, sistemin davranış repertuvarının sınırlı olmasıdır. Sistem yalnızca kendisine öğretilmiş durumları “bilir”. Bilmediği bir durumla karşılaştığında ise ya hata üretir ya da anlamsız bir davranış sergiler. Bu da sistemin gerçek dünya karşısında güvenilirliğini ciddi şekilde düşürür.

Anlamanın Yanılsaması

Dışarıdan bakıldığında tepki veren sistemler oldukça akıllı görünebilir. Bir robotun engeli algılayıp durması, sonra yön değiştirip devam etmesi, insana bilinçli bir karar süreci gibi görünebilir. Ancak bu yalnızca bir yanılsamadır.

Gerçekte sistem hiçbir şeyi “anlamaz”. Ne engelin ne olduğunu bilir, ne tehlikeyi kavrar, ne de bir karar süreci yürütür. Sadece önceden yazılmış bir kuralı uygular:

Eğer mesafe küçükse dur, değilse devam et.

Bu nedenle bu tür sistemlerde zekâ yoktur; yalnızca iyi tasarlanmış tepkiler vardır. Davranış, anlamdan değil, doğrudan kural uygulamasından doğar.

Kırılganlığın Kaynağı

Komut tabanlı sistemlerin kırılgan olmasının temel nedeni, dünyayı statik ve tamamen öngörülebilir varsaymalarıdır. Yani tüm durumların önceden tanımlanabileceği ve her senaryonun bir kurala indirgenebileceği kabul edilir. Bu varsayım mühendislik açısından cazip olsa da gerçek dünya ile uyumlu değildir.

Gerçek dünyada belirsizlik temel bir özelliktir. Küçük bir değişiklik bile sistemin davranışını tamamen bozabilir. Bu nedenle bu tür sistemler yalnızca kontrollü ortamlarda başarılıdır. Fabrika hatları, sabit üretim süreçleri veya önceden tanımlı görevler için uygundur; ancak açık ve dinamik ortamlarda başarısız olma eğilimindedir.

Robotik Düşünmenin Sınırı

Tepki veren sistemler robotik tarihinin önemli bir aşamasını temsil eder. Çünkü birçok erken dönem sistem bu mantıkla geliştirilmiştir ve hâlâ bazı uygulamalarda kullanılmaktadır. Ancak zamanla bu yaklaşımın sınırları netleşmiştir.

Robotik ilerledikçe şu temel gerçek ortaya çıkmıştır: davranış yalnızca kurallardan üretilemez. Gerçek anlamda akıllı sistemler, yalnızca komutlara değil, öğrenmeye ve uyum sağlamaya da ihtiyaç duyar.

Temel Sonuç

Komutlarla çalışan sistemler bize önemli bir gerçeği öğretir: bir sistemin çalışıyor olması, onun akıllı olduğu anlamına gelmez. Bu sistemler esnek değildir, yeni durumlara uyum sağlayamaz ve önceden tanımlanmamış senaryolarda başarısız olur.

Ancak aynı zamanda önemli bir başlangıç noktasıdırlar. Çünkü robotik düşünmenin ilk adımı, bu sınırlamaları fark etmektir.

Ve bundan sonraki adım nettir: kurallardan davranışa geçmek.

1.4 Gerçek Dünya Neden Zordur?

Robotik sistemler tasarlanırken çoğu zaman örtük bir varsayım yapılır: Dünya düzenlidir, ölçülebilirdir ve yeterince iyi modellenebilir. Ancak bu varsayım, yalnızca kontrollü laboratuvar ortamlarında geçerlidir. Gerçek dünyaya çıktığınızda bu düzen hızla çözülmeye başlar.

Gerçek dünya:

- Gürültülüdür
- Belirsizdir
- Tam tanımlı değildir

Bu üç özellik, robotik sistemlerin neden zorlandığını anlamak için temel anahtarlardır.

Gürültü: Saf Bilginin Olmaması

Gerçek dünyada hiçbir ölçüm "temiz" değildir. Bir sensör veri üretirken yalnızca doğru bilgiyi vermez; aynı zamanda hatalar, sapmalar ve rastgelelik de üretir. Bu duruma gürültü denir.

Örneğin bir mesafe sensörü, bir nesnenin 15 cm uzaklıkta olduğunu söyleyebilir. Ancak bu değer aslında 14.2, 15.8 veya 16.1 olabilir. Sensör her ölçümde biraz farklı sonuç üretir.

Bu küçük farklar ilk bakışta önemsiz gibi görünür. Ancak robotik sistemlerde bu tür küçük hatalar zamanla büyür ve sistemin davranışını tamamen değiştirebilir.

Dolayısıyla robot, "kesin bilgi" ile değil, "yaklaşık bilgi" ile çalışmak zorundadır.

Belirsizlik: Geleceğin Öngörülemezliği

Gerçek dünyanın ikinci temel özelliği belirsizliktir. Belirsizlik, bir sistemin gelecekte ne olacağını kesin olarak bilememesidir.

Bir robot için bu şu anlama gelir:

- Bir engelin ne zaman ortaya çıkacağı bilinemez
- Ortam koşulları değişebilir
- Diğer nesnelerin davranışı tahmin edilemez

Örneğin sabah boş olan bir alan, öğleden sonra tamamen dolu olabilir. Bir yol açıkken bir anda kapanabilir. Bir nesne yer değiştirebilir.

Bu durum, robotik sistemlerin en büyük zorluklarından biridir. Çünkü klasik sistemler genellikle sabit varsayımlar üzerine kuruludur. Oysa gerçek dünya sürekli değişir.

Tam Tanımlı Olmayan Sistemler

Üçüncü kritik özellik, dünyanın tam tanımlı olmamasıdır. Bir problem tanımlandığında genellikle tüm değişkenler açıkça belirtilir. Ancak gerçek dünya problemleri böyle değildir.

Bir robotun karşılaştığı durumlar:

- Eksik bilgi içerir
- Net sınırları yoktur
- Sürekli değişir

Örneğin bir robot “engelden kaç” görevi aldığında, “engel” kavramı bile net değildir. Engel nedir? Hangi boyutlar engel sayılır? Hareket eden bir nesne engel midir? Bu soruların net cevapları yoktur. Çünkü dünya, matematiksel bir problem gibi önceden tamamen tanımlanamaz.

“Her Durumu Önceden Yazayım” Yanılgısı

Robotik öğrenmeye başlayan birçok kişi şu yaklaşımı benimser:

“Her durumu önceden yazarsam sistem kusursuz çalışır.”

Bu düşünce ilk bakışta mantıklı görünür. Çünkü kontrol bizdedir ve sistemin nasıl davranacağını kesin olarak belirleyebiliriz. Ancak bu yaklaşımın temel bir problemi vardır: Gerçek dünyadaki durumların sayısı sınırsızdır.

Örneğin bir mobil robotu düşünelim:

- Zemin pürüzlü olabilir
- Işık değişebilir
- Engel sayısı değişebilir
- Sensörler farklı hata seviyelerinde çalışabilir

Bu durumların her biri için ayrı bir kural yazmak gerekir. Ancak bu kombinasyonlar hızla patlar. Çok kısa sürede yönetilemez bir karmaşıklık ortaya çıkar.

Tüm Durumları Yazmak Neden İmkânsızdır?

Matematiksel olarak bakıldığında, gerçek dünya kombinatoryal olarak büyüyen bir problem alanıdır. Her değişken yeni bir durum yaratır ve bu durumlar birbirleriyle çarpılarak artar. Bu şu anlama gelir:

- 10 değişken → yüzlerce durum
- 20 değişken → binlerce durum
- 50 değişken → pratikte sonsuz durum

> Bu nedenle tüm olası durumları kodlamak mümkün değildir. Sistem ne kadar detaylı yazılırsa yazılsın, her zaman “yazılmamış bir durum” ortaya çıkacaktır. İşte bu nokta kritik bir kırılmadır.

Kırılma Noktası: Robotik Değişir

Tüm bu zorluklar bir araya geldiğinde şu sonuç ortaya çıkar:

Klasik yaklaşım yetersizdir.

Çünkü robotik artık sadece kurallarla yönetilemez bir alan haline gelir. Sistemler, yalnızca önceden yazılmış senaryolara göre değil, yeni durumlara uyum sağlayacak şekilde tasarlanmak zorundadır.

Bu, robotik düşüncenin dönüşüm noktasıdır.

Artık mesele şudur:

- Her durumu yazmak değil
- Her durumu öğrenebilen sistemler tasarlamak

Bu nedenle modern robotik, belirsizliği bir hata değil, bir gerçeklik olarak kabul eder. Sistemler bu belirsizlikle baş edebilmek için daha esnek, daha öğrenebilir ve daha uyumlu hale getirilir.

Gerçek dünya, düzenli bir laboratuvar ortamı değildir. Gürültü, belirsizlik ve eksik tanım onun doğal yapısıdır. Bu nedenle sabit kurallara dayalı sistemler, bu dünyanın karmaşıklığı karşısında yetersiz kalır. Robotik tam da bu noktada değişir:

Kurallardan uyuma,
komutlardan davranışa geçilir.

Ve bu geçiş, modern robotik düşüncenin başlangıcıdır.

1.5 Robotik Düşünmenin Doğuşu

Robotik tarihine geriye dönüp bakıldığında, uzun bir süre boyunca temel yaklaşım oldukça basitti: bir makineye ne yapması gerektiğini söylemek. Bu yaklaşım, mühendislik açısından son derece mantıklıydı. Çünkü kontrol edilebilir sistemler üretmek, öngörülebilir sonuçlar elde etmek ve hataları minimize etmek her zaman birincil hedef olmuştur.

Ancak bu yaklaşımın içinde görünmeyen bir sınırlılık vardı. Sistemler ne kadar karmaşıklaşırsa karmaşıklaşsın, temel mantık aynı kalıyordu: “Eğer bu olursa, şunu yap.” Bu yapı, küçük ve iyi tanımlanmış problemler için oldukça etkiliydi. Fakat dünya büyüdükçe, sistemler gerçek dünyaya yaklaştıkça ve belirsizlik arttıkça bu yaklaşımın sınırları görünür hale geldi.

İşte bu noktada modern robotik düşünce ortaya çıktı. Bu düşüncenin merkezinde tek bir cümle vardır:

Robotu programlama, robotu eğit.

Bu cümle ilk bakışta basit bir slogan gibi görünebilir. Ancak aslında robotik ve yapay zekâ tarihinde yaşanan en büyük paradigma değişimlerinden birini ifade eder. Çünkü klasik yaklaşımda mühendis, robotun her davranışını tek tek tanımlamak zorundaydı. Robot hangi durumda ne yapacaksa, bunun kuralları önceden yazılırdı. Başka bir deyişle kontrol edilen şey doğrudan davranışın kendisiydi.

Yeni yaklaşım ise tamamen farklıdır. Artık mühendis robota her hareketi tek tek öğretmez; onun öğrenebileceği ortamı ve mekanizmayı tasarlar. Böylece kontrol edilen şey “davranış” değil,

davranışın nasıl oluşacağıdır. Robot deneyimlerden öğrenir, örüntüleri keşfeder ve zaman içinde kendi karar yapısını geliştirir.

Bu dönüşümün temelinde veri vardır. Çünkü modern robotik sistemler, dünyayı sabit kurallarla değil, örneklerle anlamaya çalışır. Bir çocuğun yürümeyi deneme-yanılma yoluyla öğrenmesi gibi, robot da çevreyle etkileşim içinde beceri kazanır. Bu nedenle mühendislik anlayışı da değişmektedir. Artık mesele yalnızca makine tasarlamak değil; öğrenen, uyum sağlayan ve belirsizlik altında karar verebilen sistemler geliştirmektir.

Komut Vermekten Öğretmeye Geçiş

Klasik yaklaşımda robotik mühendisinin görevi nettir: Sistemin her olası durumda ne yapacağını tek tek tanımlamak. Bu, bir anlamda çok ayrıntılı bir talimat kitabı yazmak gibidir. Robot, bu kitabı okur ve her durumda doğru sayfayı uygular. Modern yaklaşım ise farklı bir soru sorar:

"Bu robotu nasıl öğrenebilir hale getiririm?"

Bu sorunun cevabı, robotik düşünmenin doğuşunu temsil eder. Artık mühendis, robotun davranışlarını tek tek yazmaz. Bunun yerine, robotun kendi davranışlarını geliştirebileceği bir yapı tasarlar. Bu değişim, kontrol fikrini kökten değiştirir. Çünkü artık sistemin her adımı önceden belirlenmez; sistem, deneyim yoluyla kendi davranışını oluşturur.

Davranışın Yerine Öğrenme

Robotik düşünmenin en önemli kırılmalarından biri, "davranış" kavramının yeniden tanımlanmasıdır. Klasik sistemlerde davranış, doğrudan yazılan bir sonuçtur. Modern sistemlerde ise davranış, öğrenme sürecinin bir çıktısıdır.

Bu farkı anlamak kritik önemdedir. Çünkü:

- Klasik sistemlerde davranış **tasarlanır**
- Modern sistemlerde davranış **keşfedilir**

Bir robot artık sadece verilen komutları uygulayan bir varlık değildir. O, çevresiyle etkileşime giren, deneyim toplayan ve bu deneyimlerden yeni davranışlar üreten bir sistemdir.

Bu noktada öğrenme, robotik sistemin merkezine yerleşir.

"Nasıl Öğreneceğini Tasarlamak" Ne Demektir?

Modern robotikte en önemli soru artık şudur:

"Robot ne yapacak?" değil, "Robot nasıl öğrenecek?"

Bu sorunun cevabı, bir algoritma yazmaktan çok daha derin bir tasarım problemidir. Çünkü burada amaç, robotun belirli bir durumda ne yapacağını söylemek değil, o durumu nasıl yorumlayacağını öğretmektir.

Bu yaklaşımda üç temel unsur öne çıkar:

1. Veri: Robotun deneyimlediği dünya
2. Model: Bu veriyi anlamlandıran yapı
3. Geri bildirim: Doğru ve yanlışın öğrenilmesini sağlayan mekanizma.

Bu üçlü yapı, öğrenmenin temelidir. Robot artık sadece bir komut uygulayıcısı değil, aynı zamanda bir öğrenen sistemdir.

Yapay Zekâ ile Robotik Arasındaki Kesişim

Robotu eğitmek fikri, doğrudan yapay zekâ alanıyla kesişir. Çünkü öğrenme, yapay zekânın temel konusudur. Bir sistemin deneyimlerden genelleme yapabilmesi, sadece robotik değil, aynı zamanda makine öğrenmesi problemidir.

Bu nedenle modern robotik, iki alanın birleşiminde ortaya çıkar:

- Fiziksel dünya ile etkileşim (robotik)
- Öğrenme ve genelleme (yapay zekâ)

Bu birleşim, robotik sistemleri tamamen yeni bir seviyeye taşır. Artık robotlar sadece programlanan makineler değil, aynı zamanda öğrenen varlıklar haline gelir.

Bu dönüşüm, teknolojik olduğu kadar kavramsaldır. Çünkü artık sistemin davranışı sabit değildir; zaman içinde değişebilir, gelişebilir ve uyum sağlayabilir.

Kural Tabanlı Sistemlerden Öğrenen Sistemlere

Klasik robotik sistemlerde kurallar sabittir. Bir durum için bir kural yazılır ve bu kural her zaman aynı şekilde çalışır. Bu yaklaşım deterministiktir.

Ancak öğrenen sistemlerde durum farklıdır. Burada kurallar önceden yazılmaz; sistem kuralları veriden çıkarır. Yani davranış, dışarıdan dayatılmaz, içeriden öğrenilir.

Bu fark şu şekilde özetlenebilir:

- Klasik sistem: “Eğer A ise B yap”
- Öğrenen sistem: “A olduğunda ne yapmam gerektiğini öğren”

Bu küçük görünen fark, aslında robotik düşünmenin tamamını değiştirir.

Deneyim Kavramının Önemi

Öğrenen sistemlerde en kritik kavram deneyimdir. Bir robot, çevresiyle etkileşime girdikçe veri toplar. Bu veriler, onun gelecekteki davranışlarını şekillendirir.

Deneyim olmadan öğrenme mümkün değildir. Bu nedenle modern robotik, sadece kod yazma süreci değildir; aynı zamanda bir “deneyim tasarlama” sürecidir.

Bir robotun hangi ortamda, hangi koşullarda ve hangi geri bildirimlerle çalışacağı, onun nasıl öğreneceğini doğrudan belirler.

Belirsizlikle Baş Etmek

Öğrenme fikri aynı zamanda belirsizliği kabul eder. Klasik sistemler belirsizliği ortadan kaldırmaya çalışırken, modern sistemler onu yönetmeyi öğrenir.

Çünkü gerçek dünya belirsizdir. Bu belirsizlikten kaçmak mümkün değildir. Dolayısıyla robotik sistemler bu belirsizlikle baş edebilmelidir.

Öğrenen sistemler, bu nedenle daha esnek ve daha dayanıklıdır. Çünkü tek bir doğruya bağlı değildirler; deneyimden gelen olasılıklar üzerine inşa edilirler.

Yeni Bir Robotik Tanımı

Bu dönüşümün sonunda robotik artık şu şekilde tanımlanabilir:

> Robotik, yalnızca makinelerin nasıl çalıştığını değil, onların nasıl öğrendiğini tasarlayan disiplindir.

Bu tanım, klasik robotik anlayıştan çok farklıdır. Çünkü burada merkezde artık “komut” değil, “öğrenme” vardır.

Sonuç: Bir Düşünme Devrimi

“Robotu programlama, robotu eğit” cümlesi, yalnızca teknik bir öneri değildir. Bu cümle, robotik düşünmenin doğuşunu temsil eder.

Bu yaklaşım ile birlikte:

- Kontrol, yerini öğrenmeye bırakır
- Kurallar, yerini modellere bırakır
- Sabit sistemler, yerini uyum sağlayan sistemlere bırakır

Ve en önemlisi:

Robotik, bir mühendislik alanı olmaktan çıkıp bir **öğrenme bilimi** haline gelir.

Bu kitabın geri kalanında, bu dönüşümün nasıl gerçekleştiğini adım adım inceleyeceğiz. Çünkü artık mesele robot yapmak değil; **öğrenebilen sistemler tasarlamaktır.**

1.6 Bir Örnek: Denge Kurmak

Bir robotun dengede durmasını istediğinizi düşünün. Bu, robotik sistemlerde en sezgisel görünen ama aslında en öğretici problemlerden biridir. Çünkü denge problemi, hem fiziksel dünyanın belirsizliğini hem de kontrol sistemlerinin sınırlarını aynı anda görünür kılar. Bir robotu dik tutmak, ilk bakışta basit bir görev gibi görünür: eğildiğinde düzelt, devrilmeye yaklaştığında karşı yönde bir müdahale yap. Ancak bu basitlik hissi, yalnızca yüzeydedir.

Klasik robotik yaklaşım bu problemi oldukça doğrudan ele alır. Sistem, belirli ölçümlerle çalışır ve bu ölçümlere karşılık gelen önceden tanımlanmış tepkiler üretir. Örneğin bir açı sensörü robotun sola doğru eğildiğini tespit ettiğinde, motorlara sağa doğru bir düzeltme komutu gönderilir. Aynı şekilde robot sağa eğildiğinde bu kez sola doğru bir müdahale yapılır. Bu mantık, küçük sapmaların sürekli düzeltilmesi üzerine kuruludur.

Klasik yaklaşımın gücü, basitliğinden gelir. Sistem anlaşılırdır, uygulanması kolaydır ve belirli koşullar altında oldukça iyi çalışır. Kontrollü bir ortamda, öngörülebilir fiziksel koşullar altında bu tür bir sistem başarılı sonuçlar verebilir. Ancak bu yaklaşımın içinde önemli bir varsayım gizlidir: dünya yeterince basit ve öngörülebilirdir.

Gerçek dünya ise bu varsayımı sürekli olarak bozar. Zemin her zaman düz değildir, sürtünme sabit değildir, sensör ölçümleri gürültülüdür ve robotun ağırlık merkezi zamanla değişebilir. Bu tür değişkenler, basit kural tabanlı sistemlerin sınırlarını hızla görünür hale getirir. Çünkü her yeni durum, yeni bir kural gerektirir ve bu kuralların sayısı arttıkça sistemin karmaşıklığı kontrol edilemez hale gelir.

Klasik yaklaşımın temel problemi şudur: davranış doğrudan yazılır. Yani robotun nasıl tepki vereceği, her olası durumda açıkça tanımlanır. Bu durum, sistemi esnek olmaktan uzaklaştırır. Çünkü sistem yalnızca önceden bilinen durumlara yanıt verebilir. Bilinmeyen bir durumda ise ya hatalı davranır ya da tamamen başarısız olur.

İşte bu noktada modern yaklaşım devreye girer ve problem tamamen farklı bir şekilde ele alınır.

Modern yaklaşımda amaç, robotun her duruma karşı ne yapacağını yazmak değildir. Bunun yerine robotun nasıl öğreneceği tasarlanır. Denge problemi artık bir “kontrol listesi” değil, bir “öğrenme süreci” haline gelir. Robotun düşmesi bir hata değil, sürecin doğal bir parçasıdır.

Bu yaklaşımda ilk adım, robotu farklı durumlarla karşı karşıya bırakmaktır. Robot tekrar tekrar dengesini kaybeder, farklı açılarda eğilir, farklı hızlarda hareket eder. Bu süreç dışarıdan bakıldığında başarısızlık gibi görünebilir. Ancak aslında sistem veri toplamaktadır. Her düşüş, her sapma ve her toparlanma bir deneyimdir.

Bu deneyimler, robotun gelecekteki davranışlarını şekillendiren temel yapı taşlarına dönüşür. Robot artık tek tek yazılmış kurallara göre değil, geçmiş deneyimlerinden öğrendiği bir model üzerinden davranır. Hangi hareketlerin dengeyi koruduğunu, hangi durumlarda devrilmenin kaçınılmaz olduğunu zamanla keşfeder.

Modern yaklaşımın en kritik yönü, hataya bakış açısını değiştirmesidir. Klasik sistemlerde hata, düzeltilmesi gereken bir durumdur. Modern sistemlerde ise hata, öğrenmenin kaynağıdır.

Robotun düşmesi, sistemin başarısızlığı değil; öğrenme sürecinin bir parçasıdır. Bu nedenle robotun defalarca düşmesine izin vermek, aslında onu daha güçlü hale getiren bir süreçtir.

Bu yaklaşım, kontrol fikrini de kökten değiştirir. Artık mühendis robotun her hareketini belirlemez. Bunun yerine, robotun davranışlarını şekillendirecek öğrenme mekanizmasını tasarlar. Bu mekanizma, hangi davranışların iyi, hangilerinin kötü olduğunu deneyim üzerinden öğrenir.

Bu iki yaklaşım arasındaki fark oldukça derindir. Klasik yaklaşımda sistem bir "uygulayıcıdır". Ona verilen komutları yerine getirir ve bu komutların dışına çıkamaz. Modern yaklaşımda ise sistem bir "öğrenen"dir. Davranışlarını sabit kurallardan değil, deneyimden üretir.

Bu farkı daha net görmek için şu karşılaştırma yapılabilir:

Klasik yaklaşımda:

- Sola eğilme → sağa düzelt
- Sağa eğilme → sola düzelt

Modern yaklaşımda ise:

- Robot farklı durumları deneyimler
- Hangi hareketlerin dengeyi koruduğunu öğrenir
- Zamanla kendi kontrol stratejisini geliştirir

Birinci yaklaşım deterministiktir; her şey önceden tanımlıdır. İkinci yaklaşım adaptiftir; sistem zamanla değişir ve gelişir.

Bu fark, yalnızca teknik bir ayrım değildir. Aynı zamanda robotik düşünmenin temel dönüşümünü temsil eder. Çünkü artık mesele bir sistemin nasıl programlandığı değil, nasıl öğrenebilir hale getirildiğidir.

Denge problemi bu dönüşümün en iyi örneklerinden biridir. Çünkü denge, sürekli değişen ve hassas bir durumdur. Küçük bir değişiklik bile büyük sonuçlar doğurabilir. Bu nedenle sabit kurallar yerine uyum sağlayabilen sistemler çok daha etkilidir.

Sonuç olarak, denge kurma problemi bize robotik düşünmenin iki temel yaklaşımını açıkça gösterir. Bir yanda her adımı önceden tanımlayan komut tabanlı sistemler vardır. Diğer yanda ise deneyim yoluyla öğrenen, hatalardan beslenen ve zamanla gelişen sistemler.

Bu iki yaklaşım arasındaki fark yalnızca teknik değildir. Bu fark, robotik alanının nasıl evrildiğini ve gelecekte nasıl şekilleneceğini belirleyen temel ayrımdır.

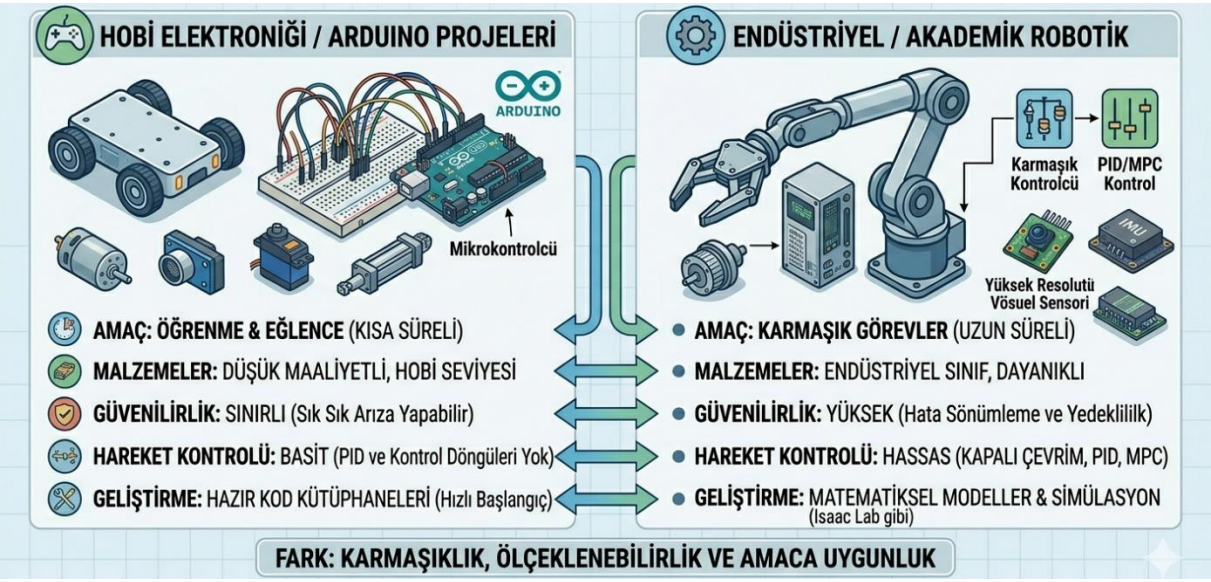
Ve bu ayrımın özeti oldukça basittir:

Birincisi komut tabanlıdır. İkincisi davranış tabanlıdır.

1.7 Robotik ≠ Arduino Projesi

Robotik denildiğinde özellikle eğitim ortamlarında akla ilk gelen araçlardan biri çoğu zaman Arduino olur. Bu durum anlaşılırdır; çünkü Arduino, düşük maliyetli olması, kolay programlanabilir yapısı ve geniş topluluk desteği sayesinde öğrenciler için erişilebilir bir giriş kapısı sunar. Bir LED'i yakıp söndürmek, bir sensörden veri okumak, bir motoru çalıştırmak ya da basit bir engelden kaçan araç yapmak, öğrenme sürecinde oldukça motive edici deneyimlerdir. Ancak tam da bu noktada ciddi bir kavramsal yanılgı ortaya çıkar: Robotik ile Arduino projeleri eşitlenir. Şekil 2'de Arduino projeleri ile endüstriyel/ akademik robotik arasında yapılan bir karşılaştırma görülmektedir.

Şekil 2. Karşılaştırma: Robotik ≠ Arduino Projesi



Bu eşitleme, teknik olarak bakıldığında eksik ve indirgemeci bir yaklaşımdır. Çünkü robotik, yalnızca elektronik devre kurmak ya da sensör verisi okumaktan ibaret değildir. Robotik, davranış üreten sistemlerin tasarımıdır. Bu davranışlar; algılama, karar verme, öğrenme ve eylem üretme süreçlerinin bir bütünüdür. Arduino ise bu sistemin yalnızca bir donanım katmanını temsil eder.

Bu nedenle şu ifade net bir şekilde kurulmalıdır:

> Arduino ile robot yapılabilir, ancak Arduino ile robotik düşünme öğrenilmiş olmaz.

Arduino'nun Gerçek Katkısı

Arduino, robotik eğitiminde önemli ve vazgeçilmez bir araçtır. Bunun nedeni, öğrencinin soyut kavramları somut bir fiziksel dünyada deneyimlemesine olanak tanımasıdır. Örneğin bir ultrasonik sensör ile mesafe ölçmek, bir sıcaklık sensörü ile çevresel değişimleri gözlemlemek veya bir servo motoru kontrol etmek, öğrenciye "algı–tepki" ilişkisini doğrudan gösterir.

Arduino'nun sağladığı temel kazanımlar şunlardır:

- Donanım bileşenlerini tanıma
- Sensörlerin nasıl veri ürettiğini öğrenme
- Aktüatörlerin (motor, servo vb.) nasıl çalıştığını görme
- Elektriksel bağlantı mantığını kavrama
- Temel programlama becerileri geliştirme

Bu yönüyle Arduino, robotik dünyasına açılan bir kapı gibidir. Ancak kapıdan içeri girmek, evin tamamını anlamak anlamına gelmez.

Eksik Kalan Katman: Davranış Üretimi

Robotik sistemleri gerçek anlamda robot yapan şey, donanım değil davranıştır. Davranış ise sadece "if-else" mantığıyla açıklanamayacak kadar karmaşıktır.

Bir robotun davranış üretebilmesi için şu sorulara cevap vermesi gerekir:

- Çevresini nasıl algılıyor?
- Bu algıyı nasıl yorumluyor?
- Hangi durumda hangi kararı veriyor?
- Zaman içinde bu kararları değiştirebiliyor mu?
- Belirsizlik altında nasıl davranıyor?

Arduino tabanlı projeler genellikle bu soruların sadece ilk seviyesine dokunur. Örneğin bir engel algılandığında geri gitmek ya da sağa dönmek gibi sabit kurallarla çalışan sistemler, "reaktif davranış" üretir. Ancak bu, gerçek robotik sistemlerde gördüğümüz adaptif ve öğrenen davranıştan oldukça uzaktır.

Gerçek robotik sistemler, çoğu zaman çok katmanlı karar yapıları içerir. Örneğin bir otonom araç yalnızca "engel var mı?" sorusuna cevap vermez; aynı zamanda:

- Engel ne kadar tehlikeli?
- Alternatif yollar nelerdir?
- Hangi hızda hareket edilmeli?
- Diğer araçların davranışları nasıl?

gibi çok boyutlu analizler yapar.

Arduino tabanlı klasik eğitim projeleri ise bu karmaşıklığın yalnızca çok küçük bir kesitini temsil eder.

Robotik = Sistem Düşüncesi

Robotik, aslında bir sistem bilimidir. Bir robotu anlamak için tek tek parçaları bilmek yeterli değildir; parçalar arasındaki ilişkileri de anlamak gerekir. Bu nedenle robotik, aşağıdaki disiplinlerin kesişim kümesinde yer alır:

- Mekanik
- Elektronik
- Bilgisayar bilimi
- Kontrol teorisi
- Yapay zekâ
- Veri analizi

Arduino ise bu büyük yapının yalnızca "elektronik giriş noktası"dır.

Bu nedenle robotik eğitimi yalnızca Arduino projeleriyle sınırlandığında, öğrenci sistemin sadece yüzeyini görür. İç yapıyı, yani sistemin nasıl düşündüğünü ve nasıl karar verdiğini göremez.

Neden Bu Karışıklık Ortaya Çıkıyor?

"Robotik = Arduino" algısının ortaya çıkmasının birkaç önemli nedeni vardır: Birincisi, eğitim materyallerinin basitleştirilmiş olmasıdır. Öğrencinin hızlı sonuç görmesi için genellikle "engel algıla–kaç" gibi basit senaryolar seçilir. Bu da robotik kavramını basitleştirir.

İkincisi, fiziksel çıktının çok görünür olmasıdır. Bir LED'in yanması ya da bir robotun hareket etmesi, öğrenme sürecini somutlaştırır. Ancak bu somutluk, derinliği gizleyebilir.

Üçüncüsü, öğretim kolaylığıdır. Karmaşık yapay zekâ algoritmalarını öğretmek yerine Arduino ile yapılan projeler daha hızlı ve yönetilebilir bir öğrenme ortamı sunar.

Ancak bu kolaylık, uzun vadede yanlış bir zihinsel model oluşturabilir.

Arduino Bir Amaç Değil, Araçtır

Bu kitapta Arduino'nun konumu bilinçli olarak "merkez" değil, "araç" olarak tanımlanmaktadır. Çünkü amaç, öğrencinin bir devreyi çalıştırması değil, bir sistemin nasıl düşündüğünü anlamasıdır.

Arduino şunu öğretir:

- Donanım nasıl çalışır
- Fiziksel dünya nasıl ölçülür
- Basit kontrol döngüleri nasıl kurulur

Ama şunu öğretmez:

👉 **Karmaşık davranış üretimi**

Karmaşık davranış üretimi; çok sayıda değişkenin aynı anda değerlendirilmesi, belirsizlik altında karar verilmesi ve zaman içinde öğrenme kapasitesinin geliştirilmesini içerir. Bu ise makine öğrenmesi, kontrol sistemleri ve yapay zekâ gibi alanların birleşimiyle mümkündür.

Robotik Eğitimin Gerçek Hedefi

Gerçek robotik eğitiminin hedefi, öğrencinin sadece "çalışan bir cihaz" üretmesi değil, "davranış üreten bir sistem" tasarlamasıdır. Bu fark oldukça kritiktir.

Bir öğrenci Arduino ile bir robot yaptığında şu soruyu sormalıdır:

"Bu robot ne yapıyor?"

Ama daha ileri seviyede şu soruyu sormalıdır:

"Bu robot neden böyle davranıyor?"

Ve en ileri seviyede:

"Bu robot kendi davranışını değiştirebilir mi?"

İşte robotik düşünme bu üçüncü soruda başlar.

Sonuç

Arduino, robotik eğitimi için güçlü bir başlangıç noktasıdır. Ancak başlangıç noktası, hedef noktası değildir. Robotik, yalnızca devre kurmak veya sensör okumak değildir; davranış tasarlama bilimidir.

Bu nedenle temel ilke şudur:

> Arduino robotik değildir, robotik öğrenmenin bir aracıdır.

Bu kitap boyunca Arduino, bir amaç olarak değil, daha büyük bir zihinsel yapıyı inşa etmek için kullanılan bir araç olarak ele alınacaktır. Çünkü gerçek robotik, bir devrenin çalışmasından değil, bir sistemin nasıl düşündüğünü anlamaktan başlar.

1.8 Robotik ≠ Yapay Zekâ (Ama Ayrı da Değil)

Modern teknoloji tartışmalarında en sık karşılaşılan kavramsal karışıklıklardan biri, robotik ile yapay zekânın birbirine eşitlenmesidir. Özellikle popüler kültürün etkisiyle "akıllı robot" ifadesi çoğu zaman hem robotik hem de yapay zekâyı aynı şeymiş gibi gösterir. Oysa bu yaklaşım hem teknik olarak eksik hem de kavramsal olarak yanıltıcıdır.

Bu nedenle sıkça şu yanlış çıkarım yapılır:

Robotik = Yapay zekâ

Bu ifade sezgisel olarak çekici görünse de doğru değildir. Robotik ve yapay zekâ birbirine bağlı iki alan olmasına rağmen, aynı şey değildir. Aralarındaki ilişki, bir kesişim ilişkisidir; özdeşlik değil. Bu ayrımı doğru anlamak, modern mühendislik sistemlerini, otonom araçları, insansı robotları ve endüstriyel otomasyonu anlamanın temelidir.

Robotik: Fiziksel Dünyanın Bilimi

Robotik, en temel anlamıyla fiziksel dünyada hareket eden sistemlerin tasarımı ve kontrolü ile ilgilenir. Robotik sistemler, gerçek dünyayla temas eden sistemlerdir. Bu nedenle robotik, "somutluk" ile ilgilidir.

Bir robotik sistem şunlarla ilgilenir:

- Mekanik yapı (kol, tekerlek, eklem)
- Sensörler (kamera, lidar, basınç sensörü)
- Aktüatörler (motorlar, servo sistemler)
- Fiziksel hareket ve kontrol
- Enerji ve kuvvet dengesi

Robotik dünyası "şeylerin nasıl hareket ettiği" sorusuna cevap verir. Bu nedenle robotik mühendisliği, fizik, mekanik ve kontrol teorisinin kesişiminde yer alır.

Örneğin bir robot kolunun bir nesneyi kaldırması tamamen robotik bir problemdir. Burada temel mesele şudur:

- Kolun hangi açıda hareket etmesi gerekir?
- Ne kadar kuvvet uygulanmalıdır?
- Denge nasıl korunur?

Bu soruların cevabı yapay zekâ gerektirmez; çoğu zaman matematiksel modelleme ve kontrol algoritmaları yeterlidir.

Yapay Zekâ: Bilgi ve Öğrenmenin Bilimi

Yapay zekâ ise fiziksel dünyadan çok, **bilgi, veri ve öğrenme süreçleri** ile ilgilidir. Yapay zekânın temel amacı, insan benzeri karar verme, öğrenme ve genelleme yeteneklerini makinelere kazandırmaktır.

Yapay zekâ şu sorularla ilgilenir:

- Sistem nasıl öğrenir?
- Veriden nasıl anlam çıkarılır?
- Belirsizlik altında nasıl karar verilir?

- Deneyimden nasıl genelleme yapılır?

Bu nedenle yapay zekâ, doğrudan fiziksel hareketten ziyade “zihinsel süreçlerin modellenmesi” ile ilgilidir.

Örneğin bir sistemin bir fotoğrafta kedi olup olmadığını anlaması, fiziksel bir problem değil, bir **algılama ve sınıflandırma problemidir**. Bu tamamen yapay zekânın alanına girer.

Makine öğrenmesi, derin öğrenme ve istatistiksel modelleme bu alanın temel araçlarıdır.

İki Alanın Temel Farkı

Robotik ve yapay zekâ arasındaki farkı en basit şekilde şöyle özetlemek mümkündür:

- Robotik: **Dünya ile etkileşim**
- Yapay zekâ: **Bilgi ile etkileşim**

Robotik sistemler fiziksel olarak hareket eder. Yapay zekâ sistemleri ise düşünsel olarak karar verir. Bir robotik sistem “nasıl hareket edeceğini” belirlerken, yapay zekâ sistemi “ne yapılması gerektiğini” belirler. Bu ayrım kritik öneme sahiptir. Çünkü modern sistemlerde bu iki katman çoğu zaman birbirine entegre edilse de, farklı problemleri çözerler.

Basit Bir Örnek: Robot Kolu

Şimdi bu ayrımı somut bir örnek üzerinden inceleyelim.

Bir robot kolu düşünelim. Bu kol bir fabrikada nesneleri kavrayıp yerleştiriyor.

Sadece Robotik Perspektif

Eğer sistem sadece robotik düzeydeyse:

- Kolun hareket yolları önceden tanımlıdır
- Sensörler yalnızca geri bildirim verir
- Hareketler sabit kurallarla kontrol edilir
- Sistem adaptif değildir

Bu durumda robot kolu, “programlandığı kadar akıllıdır”. Yeni bir durumla karşılaştığında esnek davranamaz.

Yapay Zekâ Eklenirse

Eğer aynı robot kolu yapay zekâ ile desteklenirse:

- Nesneleri tanıyabilir
- Farklı şekilleri öğrenebilir
- Yeni nesnelere adapte olabilir

- Belirsiz durumlarda karar verebilir

Örneğin daha önce görmediği bir nesneyi bile, benzer özelliklerden yola çıkarak kavrayabilir. Bu durumda sistem artık sadece mekanik bir araç değil, **öğrenen bir sistem** haline gelir.

Birleşim Noktası: Akıllı Robotik Sistemler

Modern teknolojinin en önemli gelişimi, robotik ve yapay zekânın birleşmesidir. Bu birleşimden doğan yapı şu şekilde tanımlanır:

👉 **Akıllı robotik sistem**

Bu sistemler hem fiziksel dünyayla etkileşir hem de bu etkileşimden öğrenir.

Akıllı robotik sistemler şunları yapabilir:

- Görsel veriyi analiz eder
- Çevresini algılar
- Deneyimden öğrenir
- Karar verir
- Fiziksel eylem üretir

Bu sistemler artık sadece "programlanan makineler" değildir. Aynı zamanda "öğrenen varlıklar" haline gelirler.

Neden Bu Ayrım Önemlidir?

Robotik ile yapay zekânın ayrımını doğru yapmak birkaç nedenle kritiktir. Birincisi, mühendislik tasarımını doğru anlamak için gereklidir. Bir sistemi tasarlarken hangi problemin mekanik, hangi problemin bilişsel olduğunu bilmek gerekir.

İkincisi, eğitim açısından önemlidir. Öğrenciler robotik yaptıklarını düşündüklerinde aslında çoğu zaman yalnızca kontrol sistemi öğrenirler. Yapay zekâ olmadan yapılan robotik, sınırlı bir öğrenme sağlar.

Üçüncüsü, teknolojik beklentileri yönetmek için gereklidir. "Akıllı robot" ifadesi çoğu zaman aşırı beklenti oluşturur. Oysa her robot akıllı değildir.

Robotik ve Yapay Zekâ Arasındaki Köprü

Bu iki alan arasında güçlü bir köprü vardır. Bu köprü aşağıdaki alanlardan oluşur:

- Kontrol teorisi
- Algılama sistemleri
- Bilgisayarla görme

- Robot öğrenmesi
- Otonom sistemler

Özellikle son yıllarda “robot öğrenmesi” (robot learning) alanı bu iki disiplini doğrudan birleştirmiştir. Burada robotlar, çevreyle etkileşerek öğrenirler. Örneğin bir robot kolu, deneme-yanılma yoluyla bir nesneyi nasıl daha iyi kavrayacağını öğrenebilir. Bu süreçte hem fiziksel hareket (robotik) hem de öğrenme algoritmaları (yapay zekâ) birlikte çalışır.

Günümüz Uygulamaları

Bugün kullandığımız birçok sistem bu birleşimi temsil eder:

- Otonom araçlar
- Depo robotları
- İnsansı robotlar
- Drone sistemleri
- Akıllı üretim hatları

Bu sistemlerin ortak özelliği şudur:

👉 Fiziksel dünyada hareket ederler ve aynı zamanda öğrenirler.

Bir otonom araç yalnızca bir robot değildir. Aynı zamanda çevresini analiz eden, trafik durumunu öğrenen ve karar veren bir yapay zekâ sistemidir.

Yanlış Anlamanın Kaynağı

“Robotik = Yapay zekâ” yanlış anlamasının temel nedeni, popüler kültürdür. Filmler, diziler ve medya genellikle robotları “insan gibi düşünen makineler” olarak sunar. Bu da iki alanın birbirine karışmasına yol açar.

Oysa gerçek dünyada çoğu robot:

- Çok basit kurallarla çalışır
- Sınırlı görevler yapar
- Öğrenme yeteneğine sahip değildir

Yapay zekâ ise çoğu zaman fiziksel bir bedene bile sahip değildir. Sadece veri üzerinde çalışır.

Sonuç

Robotik ve yapay zekâ aynı şey değildir. Ancak modern dünyada birbirinden ayrı da düşünülemezler. Robotik fiziksel dünyayı yönetir, yapay zekâ ise bilgi dünyasını yönetir. Bu iki alan birleştiğinde ise ortaya yeni bir mühendislik paradigması çıkar:

👉 **Akıllı robotik sistemler**

Bu sistemler artık sadece hareket eden makineler değil, aynı zamanda öğrenen ve karar veren yapılar haline gelmiştir. Bu kitap boyunca bu ayrım kritik bir temel olarak kabul edilecektir. Çünkü robotik düşünmeyi anlamak, yalnızca hareketi değil; hareketin arkasındaki zekâyı da anlamayı gerektirir.

1.9 Bu Kitapta Ne Yapacağız?

Bu kitap, yalnızca bir programlama kılavuzu ya da robotik projeler rehberi değildir. Aynı zamanda bir düşünme biçimi kazandırmayı hedefler. Çünkü modern robotik ve yapay zekâ sistemlerini anlamak, yalnızca kod yazmayı öğrenmekle değil, sistemleri katmanlı bir şekilde düşünebilmekle mümkündür.

Bugünün dünyasında robotlar artık sadece mekanik cihazlar değildir. Onlar, veriden öğrenen, karar veren ve çevresiyle etkileşime giren karmaşık sistemlerdir. Bu nedenle robotik, programlama, makine öğrenmesi ve fiziksel sistemler birbirinden ayrı alanlar olmaktan çıkmış, tek bir bütün haline gelmiştir.

Bu kitapta işte tam olarak bu bütünlüğü kuracağız.

Üç temel bileşen etrafında ilerleyeceğiz:

1. Programlama (Python)
2. Öğrenme (PyTorch)
3. Fiziksel sistemler (robotik)

Bu üçlü yapı, modern akıllı sistemlerin iskeletini oluşturur. Her biri tek başına önemli olsa da, asıl güç bu üç alanın birleşiminden doğar.

1. Programlama: Python ile Düşünmeyi Öğrenmek

Bu yolculuğun ilk adımı programlamadır ve bu adımda ana dilimiz Python olacaktır. Python yalnızca bir programlama dili değildir; aynı zamanda düşünceyi sadeleştiren bir araçtır.

Bir sistemi anlamanın en güçlü yollarından biri onu kodlamaktır. Çünkü kod yazmak, düşüncenin açık ve kesin bir forma dönüşmesidir. Belirsiz fikirler, kod yazarken zorunlu olarak netleşir.

Bu kitapta Python'u şu amaçlarla kullanacağız:

- Algoritmik düşünme geliştirmek
- Veri yapıları ile çalışmak
- Basit simülasyonlar kurmak
- Sensör verisini modellemek

- Yapay sistemlerin temel mantığını anlamak

Örneğin bir robotun hareketini düşünmek yerine, o hareketi kod olarak ifade etmek, zihinsel modeli daha net hale getirir.

Python burada bir amaç değil, bir düşünme aracıdır. Amacımız Python uzmanı olmak değil, Python üzerinden sistemleri anlamayı öğrenmektir.

Bu aşamada öğrenci şunu fark etmeye başlar:

👉 Bir robotu anlamak, aslında bir davranışı kodlamayı anlamaktır.

2. Öğrenme: PyTorch ile Yapay Zekânın Mantığını Anlamak

İkinci adımda öğrenme kavramına geçeceğiz. Burada kullanacağımız temel araç PyTorch olacaktır.

PyTorch, modern yapay zekâ sistemlerinin en yaygın kullanılan derin öğrenme kütüphanelerinden biridir. Ancak bu kitapta PyTorch'u yalnızca bir yazılım aracı olarak değil, öğrenmenin matematiksel ve kavramsal bir modeli olarak ele alacağız.

Çünkü yapay zekânın özü kod değil, öğrenmedir.

Bu bölümde şu sorulara cevap arayacağız:

- Bir makine nasıl öğrenir?
- Öğrenme süreci nasıl matematiksel olarak ifade edilir?
- Bir model nasıl hata yapar ve bu hatadan nasıl öğrenir?
- Ağırlıklar ve parametreler neyi temsil eder?

PyTorch bize şu üç temel kavramı öğretir:

- Veri
- Model
- Optimizasyon

Bu üçlü yapı, modern yapay zekânın temelidir.

Örneğin bir model, bir görüntüde kedi olup olmadığını öğrenirken binlerce örnek görür. Her örnekten sonra yaptığı hatayı azaltmaya çalışır. Bu süreç, insan öğrenmesine oldukça benzer bir yapıya sahiptir.

Bu aşamada artık programlama sadece "komut yazma" olmaktan çıkar. Programlama, "öğrenen sistem tasarlama" haline gelir.

Bu dönüşüm çok kritiktir.

Çünkü burada ilk kez şu fikri net şekilde görürüz:

👉 Sistemler sadece çalışmaz, öğrenebilir.

Bu anlayış, robotik dünyasının kapısını açar.

3. Fiziksel Sistemler: Robotik ile Gerçek Dünyaya Dokunmak

Üçüncü adımda artık soyut dünyadan çıkıp fiziksel dünyaya giriyoruz. Robotik burada devreye giriyor. Robotik, yapay zekânın ve programlamanın gerçek dünyadaki karşılığıdır. Çünkü bir robot yalnızca veri üzerinde çalışan bir model değildir; aynı zamanda hareket eden, çevreyle etkileşen fiziksel bir varlıktır.

Bu aşamada şu sorular önem kazanır:

- Bir sistem fiziksel dünyada nasıl hareket eder?
- Sensörlerden gelen veri nasıl yorumlanır?
- Karar, fiziksel eyleme nasıl dönüşür?
- Yazılım ve donanım nasıl birlikte çalışır?

Robotik sistemler, üç katmandan oluşur:

- Algılama (sensörler)
- Karar verme (yazılım / yapay zekâ)
- Eylem (motorlar / aktüatörler)

Bu üç katman birlikte çalıştığında, bir robot ortaya çıkar.

Örneğin bir robot kolu bir nesneyi kaldırdığında:

- Kamera nesneyi görür
- Yazılım nesnenin konumunu hesaplar
- Motorlar kolu hareket ettirir

Bu zincir, modern robotik sistemlerin temelidir. Bu kitapta robotik sadece mekanik bir alan olarak değil, öğrenen sistemlerin fiziksel dünyaya uygulanması olarak ele alınacaktır.

4. Üç Katmanın Birleşimi: Akıllı Sistemler

Kitabın en önemli noktası, bu üç alanın birleştiği yerdir.

- Python → düşünme ve modelleme
- PyTorch → öğrenme ve adaptasyon
- Robotik → fiziksel uygulama

Bu üçlü birleştiğinde ortaya şu yapı çıkar:

👉 **Akıllı sistem**

Akıllı sistem, yalnızca çalışan bir makine değildir. Aynı zamanda öğrenen, adapte olan ve çevresiyle etkileşen bir yapıdır. Bu noktada artık klasik mühendislik yaklaşımı yeterli değildir. Çünkü sistem artık statik değil, dinamiktir.

Örneğin bir robot sadece önceden programlanmış hareketleri yapmaz. Aynı zamanda çevresine göre davranışını değiştirebilir. Bu değişim, robotik dünyasında devrim niteliğindedir.

5. Modern Sistemler: NVIDIA Isaac Lab ile Tanışma

Kitabın ilerleyen bölümlerinde modern robotik simülasyon sistemlerine de değineceğiz. Bu sistemlerin en önemlilerinden biri NVIDIA Isaac Lab'tir.

> NVIDIA Isaac Lab, robotik sistemlerin sanal ortamda eğitilmesini ve test edilmesini sağlar. Bu, gerçek dünyaya geçmeden önce binlerce deney yapılabileceği anlamına gelir.

Bu platformun önemi şuradadır:

- Gerçek robotlar pahalıdır
- Gerçek deneyler risklidir
- Simülasyonlar hızlıdır ve güvenlidir

Isaac Lab sayesinde bir robot:

- Binlerce farklı ortamda test edilebilir
- Öğrenme algoritmaları hızla eğitilebilir
- Fiziksel hatalar önceden görülebilir

Bu yaklaşım, modern robotik geliştirme süreçlerinin temelidir. Bu kitapta Isaac Lab'i bir son nokta değil, modern robotik ekosisteminin bir örneği olarak ele alacağız.

6. Kitabın Genel Yol Haritası

Bu kitabın yapısı doğrusal bir anlatım değil, katmanlı bir öğrenme sürecidir.

İlerleyiş şu şekilde olacaktır:

1. Önce Python ile düşünmeyi öğreneceğiz
2. Sonra PyTorch ile öğrenmenin mantığını anlayacağız
3. Daha sonra bu bilgiyi robotik sistemlere uygulayacağız

4. En sonunda modern simülasyon sistemleriyle büyük resmi göreceğiz

Bu yapı bilinçli olarak seçilmiştir. Çünkü gerçek dünyadaki akıllı sistemler de aynı şekilde katmanlıdır. Hiçbir robot doğrudan "akıllı" olarak başlamaz. Önce programlanır, sonra öğrenir, sonra fiziksel dünyaya uyum sağlar.

7. Neden Bu Üçlü Yapı?

Bu üçlü yapı, modern teknolojinin temel eksenlerini temsil eder.

- Programlama olmadan sistem kurulamaz
- Öğrenme olmadan sistem adapte olamaz
- Robotik olmadan sistem fiziksel dünyada var olamaz

Bu nedenle bu üç alan birbirini tamamlar.

Tek başına Python bir yazılım dili olur.
Tek başına PyTorch bir kütüphane olur.
Tek başına robotik bir mühendislik alanı olur.

Ama birlikte olduklarında:

👉 Modern yapay zekâ sistemleri ortaya çıkar.

8. Sonuç: Bir Sistem Düşüncesine Giriş

Bu kitap, sadece teknik bilgi öğretmeyi değil, sistem düşüncesi kazandırmayı hedefler. Çünkü modern dünyada en önemli beceri, parçaları ayrı ayrı bilmek değil, onları bir bütün olarak görebilmektir.

Bu noktada okuyucu şu zihinsel dönüşümü yaşar:

- Kod yazmak → sistem düşünmek
- Model kurmak → öğrenmeyi anlamak
- Robot yapmak → davranış tasarlamak

Bu dönüşüm gerçekleştiğinde, artık yalnızca kullanıcı değil, sistem tasarımcısı olunmuş olur. Bu kitap boyunca ilerledikçe, Python'dan başlayıp PyTorch'a, oradan robotik sistemlere ve en sonunda modern simülasyon platformlarına kadar uzanan bir yolculuk tamamlanacaktır. Ve bu yolculuğun sonunda ortaya çıkan şey yalnızca teknik bilgi değil, yeni bir bakış açısı olacaktır:

1.10 Bir Zihniyet Değişimi

Bu kitabın en önemli amacı, yalnızca teknik bilgi aktarmak değildir. Python öğretmek, PyTorch kullanmayı göstermek ya da robotik sistemleri tanıtmak bu sürecin sadece yüzeyidir. Asıl hedef, çok daha derin bir yerde durur: düşünme biçimini değiştirmek.

Çünkü modern teknoloji dünyasında asıl farkı yaratan şey araçlar değil, zihniyettir. Aynı programlama dilini kullanan iki kişi tamamen farklı sonuçlara ulaşabilir. Aynı robotik kitiyle çalışan iki öğrenci, birbirinden çok farklı sistemler tasarlayabilir. Aynı yapay zekâ modelini eğiten iki araştırmacı, tamamen farklı anlayış seviyelerine ulaşabilir.

Bunun nedeni teknik bilgi eksikliği değildir. Bunun nedeni zihniyet farkıdır. Bu bölümün amacı, sizi teknik bir kullanıcı olmaktan çıkarıp sistem düşünen bir tasarımcı haline getirmektir.

Teknik Bilgi ile Zihniyet Arasındaki Fark

Teknik bilgi, öğrenilebilir ve öğretilebilir bir şeydir. Bir dili öğrenmek, bir kütüphaneyi kullanmak, bir algoritmayı uygulamak belirli bir zaman içinde edinilebilir becerilerdir.

Zihniyet ise farklıdır. Zihniyet, bir problemi nasıl gördüğünüzü belirler.

Örneğin aynı problemi iki kişi farklı şekilde ele alabilir:

- Biri problemi “kod yazma sorunu” olarak görür
- Diğeri problemi “davranış tasarlama sorunu” olarak görür

İlk yaklaşım genellikle kısa vadeli çözümler üretir. İkinci yaklaşım ise sistemin bütününü anlamaya çalışır.

Bu kitapta hedeflenen dönüşüm tam olarak budur:

👉 Kod yazan birinden davranış tasarlayan birine dönüşmek

Kod Yazmak Ne Demektir?

Kod yazmak, belirli bir problemi çözmek için bilgisayara talimat vermektir. Bu süreç genellikle şu adımlardan oluşur:

- Girdi tanımlama
- Kuralları belirleme
- Çıktıyı üretme

Bu yaklaşım deterministik sistemler için oldukça etkilidir. Örneğin bir hesap makinesi yapmak, bir veri sıralama algoritması yazmak ya da bir sensör verisini okumak kod yazma problemleridir.

Kod yazma sürecinde odak noktası şudur:

- “Ne yapmalıyım?”
- “Hangi adımları izlemeliyim?”
- “Bu problemi nasıl çözerim?”

Bu yaklaşım mühendislik açısından temel ve gereklidir. Ancak modern sistemlerin karmaşıklığı arttıkça bu yaklaşım tek başına yeterli olmamaya başlar.

Çünkü gerçek dünya deterministik değildir.

Davranış Tasarlamak Ne Demektir?

Davranış tasarlamak, kod yazmaktan çok daha geniş bir kavramdır. Burada amaç sadece bir problemi çözmek değil, bir sistemin nasıl davranacağını tanımlamaktır.

Davranış tasarlarken şu sorular önem kazanır:

- Sistem çevreyi nasıl algılar?
- Algıladığı bilgiyi nasıl yorumlar?
- Hangi durumda hangi tepkiyi verir?
- Zaman içinde davranışını değiştirebilir mi?

Bu sorular artık sadece algoritma değil, sistem tasarımı sorularıdır.

Davranış tasarlamak, bir robotun yalnızca hareket etmesini değil, anlamlı hareket etmesini sağlamaktır.

Örneğin:

- Bir robot engel gördüğünde durabilir (basit kod)
- Ama aynı robot engelin türüne göre farklı kararlar verebilir (davranış tasarımı)

Bu ikinci durum, artık yalnızca programlama değil, karar verme sistemidir.

Kod ve Davranış Arasındaki Fark

Kod ve davranış arasındaki farkı anlamak, bu kitabın en kritik zihinsel eşiklerinden biridir.

Kod:

- Statiktir
- Açık kurallara dayanır
- Önceden tanımlanmıştır
- Değişkenlik sınırlıdır

Davranış:

- Dinamiktir
- Çevreye bağlıdır
- Öğrenebilir
- Uyarlanabilir

Kod yazan kişi bir sistemin “ne yapacağını” tanımlar. Davranış tasarlayan kişi ise sistemin “nasıl düşüneceğini” belirler. Bu fark küçük gibi görünse de, aslında tüm modern teknolojinin temel ayrımıdır.

Neden Bu Ayrım Önemlidir?

Çünkü dünya artık sabit kurallarla çalışan sistemlerden oluşmuyor. Modern sistemler:

- Belirsizlik içerir
- Sürekli değişir
- Veri ile şekillenir
- Öğrenerek gelişir

Bu ortamda sadece kod yazabilen bir yaklaşım yetersiz kalır.

Örneğin bir yapay zekâ modeli düşünelim. Bu modelin amacı sadece bir çıktıyı üretmek değildir. Aynı zamanda:

- Yeni verilerden öğrenmek
- Hatalarını düzeltmek
- Farklı durumlara uyum sağlamak

Bu süreçler kod yazma değil, davranış tasarlama sürecidir.

Robotik ve Yapay Zekâ Perspektifi

Robotik sistemlerde de aynı dönüşüm geçerlidir.

Eski yaklaşım:
- Robot = önceden programlanmış makine

Yeni yaklaşım:
- Robot = öğrenen ve karar veren sistem

Eski robotlar yalnızca kodu çalıştırırdı. Yeni nesil robotlar ise çevreyi analiz eder ve kendi davranışlarını oluşturur. Bu nedenle modern robotik artık sadece mühendislik değil, aynı zamanda bilişsel sistem tasarımıdır.

Öğrenen Sistemler ve Davranış

Makine öğrenmesi bu dönüşümün merkezinde yer alır. Çünkü öğrenen sistemler artık sabit kurallara bağlı değildir.

Bir model:

- Veriden öğrenir

- Hatalarından gelişir
- Zamanla davranışını değiştirir

Bu süreçte kod sabit kalır, ama davranış değişir.

İşte bu noktada kritik bir gerçek ortaya çıkar:

👉 Modern sistemlerde önemli olan kod değil, ortaya çıkan davranıştır.

Zihniyet Değişiminin Aşamaları

Bu kitap boyunca yaşanacak zihinsel dönüşüm üç aşamada gerçekleşir:

1. **Kod Odaklı Düşünme**
   Sistemler yalnızca yazılan komutlardan ibarettir.
2. **Model Odaklı Düşünme**
   Sistemler veri ile çalışan öğrenen yapılardır.
3. **Davranış Odaklı Düşünme**
   Sistemler çevreyle etkileşen ve adaptif varlıklardır.

Son aşama, modern mühendisliğin gerçek seviyesidir.

Geleceğin Becerisi

Bu noktada en önemli soru şudur:

Gelecek hangi tür düşünme biçimine aittir?

Cevap nettir:

👉 Davranış tasarlamayı bilenlere

Çünkü gelecekte sistemler:

- Daha otonom olacak
- Daha öğrenen olacak
- Daha az öngörülebilir olacak
- Daha fazla adaptif olacak

Bu tür sistemleri anlamak için yalnızca kod yazmak yeterli değildir. Sistemin nasıl davrandığını anlamak gerekir.

Basit Bir Karşılaştırma

İki kişi düşünelim:

Birinci kişi:

- Kod yazar
- Problemleri algoritma olarak görür
- Kurallarla çalışır

İkinci kişi:

- Sistem tasarlar
- Davranış modelleri kurar
- Öğrenmeyi analiz eder

Kısa vadede bu iki kişi benzer sonuçlar üretebilir. Ancak uzun vadede fark büyür.

Çünkü ikinci kişi, sistemler değiştiğinde de uyum sağlayabilir.

Sonuç: Asıl Dönüşüm

Bu bölümün temel mesajı teknik değildir. Bu bölüm bir yön değişimidir.

Kod yazmak önemlidir, ancak yeterli değildir. Asıl mesele, sistemlerin nasıl davrandığını anlamaktır.

Bu kitap boyunca öğreneceğiniz her şey, sizi bu zihniyet değişimine hazırlamak içindir.

Sonuç olarak:

- Kod yazmak bir beceridir
- Davranış tasarlamak bir düşünme biçimidir

Ve bu iki şey aynı değildir.

Gelecek, yalnızca birinciyi bilenlere değil, ikinciyi anlayanlara ait olacaktır.

1.11 Bölüm Özeti

Bu bölümde, robotik ve akıllı sistemlere giriş niteliğinde çok temel ama kritik bir zihinsel çerçeve kurduk. Amaç yalnızca teknik kavramları listelemek değildi; robotik düşünmenin nasıl şekillendiğini ve neden klasik “makine” anlayışının artık yetersiz kaldığını göstermekti. Çünkü modern dünyada robotlar artık sadece hareket eden cihazlar değil, algılayan, karar veren ve öğrenen sistemlerdir.

Bu noktaya gelindiğinde, robotu tanımlama biçimimiz köklü bir değişime uğrar. Artık robotu “metal parçalarla yapılmış ve motorlarla hareket eden bir makine” olarak görmek yeterli değildir. Robot, çevresini anlayan ve bu anlayışa göre davranış üreten bir sistemdir. Bu dönüşüm, bu bölümün en temel zihinsel kazanımıdır.

Robot yalnızca hareket eden bir makine değildir

Geleneksel bakış açısında robot, belirli komutları yerine getiren mekanik bir araçtır. Ona verilen talimatları uygular ve bu talimatların dışına çıkmaz. Örneğin bir endüstriyel robot kolu, belirli bir montaj işlemini tekrar tekrar aynı şekilde yapabilir. Bu tür sistemler, uzun yıllar boyunca üretim dünyasının temelini oluşturmuştur.

Ancak bu yaklaşımın sınırları vardır. Çünkü böyle bir robot, çevresindeki değişikliklere uyum sağlayamaz. Ortam değiştiğinde ya da beklenmeyen bir durum oluştuğunda davranışını değiştiremez. Bu nedenle klasik robot tanımı, modern sistemleri açıklamakta yetersiz kalır.

Bu bölümde öğrendiğimiz ilk önemli fikir şudur:

👉 Robot sadece hareket eden bir makine değildir.

Robot, hareket eden bir sistem olmanın ötesinde, çevreyle etkileşen ve bu etkileşimi işleyen bir yapıdır.

Robotik: algı – karar – eylem üçlüsü

Robotik sistemleri anlamanın en temel yolu onları üç bileşen üzerinden düşünmektir:

- Algı
- Karar
- Eylem

Bu üçlü yapı, robotik sistemlerin iskeletini oluşturur.

Algı, robotun çevresinden veri toplamasıdır. Bu veri sensörler aracılığıyla elde edilir. Kamera görüntüleri, mesafe ölçümleri, sıcaklık verileri veya basınç bilgileri algı katmanının parçalarıdır. Algı olmadan robot, çevresinde ne olup bittiğini bilemez.

Karar, algılanan verinin yorumlanmasıdır. Bu aşamada sistem, elde ettiği bilgiyi anlamlandırır ve ne yapması gerektiğine karar verir. Basit sistemlerde bu karar kurallara dayanır. Gelişmiş sistemlerde ise öğrenme ve yapay zekâ devreye girer.

Eylem ise kararın fiziksel dünyaya yansımasıdır. Motorların çalışması, robot kolunun hareket etmesi veya bir aracın yön değiştirmesi bu aşamada gerçekleşir.

Bu üç aşama birlikte çalıştığında robotik sistem tamamlanır. Eğer bu üç bileşenden biri eksikse, sistem ya sadece bir sensör ağına ya da sadece bir aktüatör sistemine dönüşür. Robotik ise ancak bu üçlü yapı birlikte olduğunda anlam kazanır.

Komut tabanlı sistemlerin sınırları

Bu bölümde ayrıca komut tabanlı sistemlerin sınırlarını da gördük. Komut tabanlı sistemler, önceden tanımlanmış kurallara göre çalışan yapılardır. Bu sistemlerde her durum için bir komut vardır ve sistem yalnızca bu komutları uygular.

Bu yaklaşım basit problemler için oldukça etkilidir. Ancak gerçek dünya problemleri karmaşık, değişken ve öngörülemezdir. Bu nedenle yalnızca komutlara dayalı sistemler bu karmaşıklığı yönetemez.

Örneğin:

- Ortam değiştiğinde sistem uyum sağlayamaz
- Beklenmeyen durumlarda hata verir
- Yeni durumlara genelleme yapamaz

Bu sınırlamalar, modern robotik sistemlerin neden daha gelişmiş yaklaşımlara ihtiyaç duyduğunu açıklar.

Bu noktada önemli bir fark ortaya çıkar:

👉 Komut tabanlı sistemler "ne yapılacağını" bilir, ancak "neden ve ne zaman değişmesi gerektiğini" bilemez.

Modern robotik öğrenmeye dayanır

Günümüz robotik sistemlerinin en önemli özelliği öğrenme kapasitesidir. Artık robotlar yalnızca önceden programlanan davranışları sergilemez; aynı zamanda deneyimlerinden öğrenir.

Öğrenen robotik sistemler:

- Veriden model oluşturur
- Hatalarını azaltır
- Yeni durumlara uyum sağlar
- Zaman içinde performansını geliştirir

Bu dönüşüm, robotik alanını köklü şekilde değiştirmiştir. Artık robotik, sadece mekanik mühendislik değil, aynı zamanda veri bilimi ve yapay zekâ ile iç içe geçmiş bir alandır.

Öğrenme sayesinde robotlar, sabit sistemlerden dinamik sistemlere dönüşür. Bu da onları gerçek dünyadaki belirsizliklere karşı daha dayanıklı hale getirir.

Robotik = donanım + yazılım + öğrenme

Bu bölümde ulaşılan en önemli sonuçlardan biri robotik sistemlerin yapısal tanımıdır. Modern robotik artık üç temel bileşenin birleşimi olarak görülmelidir:

- Donanım
- Yazılım
- Öğrenme

Donanım, robotun fiziksel kısmıdır. Motorlar, sensörler, gövde yapısı ve mekanik bileşenler bu katmanı oluşturur.

Yazılım, robotun beyni gibi çalışır. Veriyi işler, algoritmaları çalıştırır ve sistemin nasıl davranacağını belirler.

Öğrenme ise sistemin zaman içinde gelişmesini sağlar. Bu katman, robotu statik bir makineden dinamik bir sisteme dönüştürür.

Bu üç bileşen birlikte çalıştığında gerçek anlamda modern bir robotik sistem ortaya çıkar. Eğer bu üçlüden biri eksik olursa sistem ya yalnızca bir makine ya da yalnızca bir yazılım haline gelir.

Genel değerlendirme

Bu bölümün temel amacı, robotik hakkında temel ama doğru bir zihinsel model oluşturmaktır. Robotik, yalnızca "robot yapmak" değildir. Robotik, algılayan, karar veren ve eylem üreten sistemleri anlamaktır.

Bu çerçevede öğrendiğimiz temel noktalar şunlardır:

- Robot, sadece hareket eden bir makine değildir
- Robotik, algı–karar–eylem üçlüsünden oluşur
- Komut tabanlı sistemler sınırlıdır ve değişime uyum sağlayamaz
- Modern robotik sistemler öğrenmeye dayanır
- Robotik, donanım, yazılım ve öğrenmenin birleşimidir

Bu beş temel fikir, kitabın ilerleyen bölümlerinde ele alınacak tüm konuların zihinsel temelini oluşturur. Çünkü artık robotik sadece bir mühendislik alanı değil, aynı zamanda bir düşünme biçimidir.

Bu kitap boyunca sık sık aynı temel soruya geri döneceğiz: "Bu sistemi ben mi kontrol ediyorum, yoksa sistem kendi davranışını mı üretiyor?" Bu soru, yalnızca teknik bir kontrol meselesi değildir; aynı zamanda bir düşünme biçiminin sınırlarını belirler. Çünkü klasik mühendislik yaklaşımında sistemler doğrudan insan tarafından kontrol edilir, her adım önceden tanımlanır ve her sonuç bir komutun karşılığıdır. Ancak modern sistemlerde durum giderek değişir. Sistemler yalnızca verilen komutları uygulayan yapılar olmaktan çıkar, çevreden gelen veriyi yorumlayan, bu veriye göre karar üreten ve zaman içinde davranışlarını değiştirebilen yapılara dönüşür.

Eğer bu soruya verdiğiniz cevap giderek "sistemi ben kontrol ediyorum"dan "sistem kendi davranışını üretiyor" yönüne kayıyorsa, artık robotik düşünmeye başlamışsınız demektir. Çünkü robotik düşünme, kontrolün tamamen insanda olduğu bir dünyadan, davranışın sistem içinde ortaya çıktığı bir dünyaya geçişi ifade eder. Bu geçişte önemli olan, kontrolü kaybetmek değil, kontrolün nasıl yeniden tanımlandığını anlamaktır. Modern robotik ve yapay zekâ

sistemlerinde asıl beceri, her adımı tek tek yönetmek değil, sistemin hangi koşullarda nasıl davranacağını tasarlayabilmektir.

1.12 Düşünme Soruları ve Cevapları

1. Bir sistemi "akıllı robot" yapan şey donanımı mı, yoksa davranış üretme kapasitesi midir?

Cevap:
Akıllı robotu tanımlayan şey donanım değil, davranış üretme kapasitesidir. Donanım yalnızca fiziksel eylemi mümkün kılar; ancak "akıllılık", çevreye uygun, esnek ve öğrenebilen davranışların ortaya çıkmasıyla ilgilidir. Aynı donanım, farklı yazılım ve öğrenme sistemleriyle tamamen farklı zeka seviyelerinde davranabilir. Bu nedenle asıl belirleyici olan, sistemin karar verme ve öğrenme yeteneğidir.

2. Algı–karar–eylem döngüsünün hangi kısmı başarısız olursa robotik sistem tamamen işlevsiz hale gelir?

Cevap:
Üç bileşen de kritik olsa da algı (perception) katmanındaki ciddi bir hata sistemi temelden işlevsiz hale getirir. Çünkü yanlış algı, yanlış karar ve yanlış eylem zincirini başlatır. Ancak bazı durumlarda karar mekanizması hatalıysa sistem doğru veriyi bile yanlış kullanabilir. Bu nedenle en kritik nokta, algı ile karar arasındaki tutarlılıktır.

3. Komut tabanlı bir robot ile öğrenen bir robot arasındaki en temel mimari fark nedir?

Cevap:
Komut tabanlı robotlar sabit kurallarla çalışır; davranışları önceden programlanmıştır. Öğrenen robotlar ise deneyimden politika üretir. Birincisi deterministik ve kapalıdır, ikincisi adaptif ve dinamiktir. Öğrenen robot, yeni durumlara uyum sağlayabilirken komut tabanlı robot yalnızca tanımlı senaryolarda çalışır.

4. Bir robotun çevresini "anlaması" teknik olarak ne anlama gelir?

Cevap:
Teknik olarak "anlamak", çevreden gelen sensör verilerinin anlamlı temsillere dönüştürülmesi demektir. Bu, nesne tanıma, konum tahmini, durum kestirimi ve bağlam oluşturmayı içerir. Yani robot çevreyi doğrudan "bilmez", onu matematiksel bir model olarak temsil eder ve bu model üzerinden karar verir.

5. Öğrenme kapasitesi olmayan bir robot, değişen bir ortamda ne tür hatalar üretir?

Cevap:
Öğrenme kapasitesi olmayan robotlar adaptasyon yapamaz. Ortam değiştiğinde eski kuralları uygulamaya devam ederler. Bu durum şu hatalara yol açar: yanlış hareketler, hedefe ulaşamama, çarpışmalar ve verimsizlik. Özellikle belirsiz ve dinamik ortamlarda sistem hızla işlevsiz hale gelir.

6. Yapay zekâ olmadan geliştirilen robotik sistemler hangi tür problemleri çözebilir, hangilerini çözemez?

Cevap:
Yapay zekâ olmadan robotlar yalnızca önceden tanımlanmış, statik ve öngörülebilir problemleri çözebilir. Örneğin sabit bir montaj hattında tekrar eden işlemler mümkündür. Ancak değişken ortamlar, nesne tanıma, öğrenme gerektiren görevler ve belirsizlik içeren durumlar çözülemez.

7. Bir robotun davranışı yazılım tarafından mı belirlenir, yoksa veriden mi ortaya çıkar? Bu iki yaklaşım nasıl birleşir?

Cevap:
Modern sistemlerde davranış hem yazılım tasarımı hem de veriden öğrenme ile oluşur. Başlangıçta mimari yazılım tarafından belirlenir (model yapısı, ödül fonksiyonu), ancak davranışın ayrıntıları veriden öğrenilir. Yani yazılım "çerçeveyi", veri ise "içeriği" üretir.

8. Fiziksel bir robotun karar verme sürecinde sensör verisinin rolü nedir ve bu veri nasıl "anlama" dönüşür?

Cevap:
Sensör verisi robotun dünyaya açılan tek kapısıdır. Kamera, lidar veya dokunma sensörlerinden gelen ham veri doğrudan anlam taşımaz. Bu veriler işlenerek özelliklere, sonra durum temsillerine dönüştürülür. "Anlama", bu temsilin karar mekanizmasıyla ilişkilendirilmesiyle oluşur.

9. Robotik sistemlerde "genelleme" kavramı neden kritik bir yetenektir ve nasıl elde edilir?

Cevap:
Genelleme, bir robotun daha önce görmediği durumlarda da doğru davranabilmesidir. Bu kritik bir yetenektir çünkü gerçek dünya sonsuz varyasyon içerir. Genelleme, çeşitlendirilmiş veri, simülasyon (domain randomization), düzenlileştirme teknikleri ve güçlü model mimarileri ile elde edilir.

10. Gelecekte robotların tasarımında en önemli beceri kod yazmak mı olacak, yoksa davranış modellemek mi?

Cevap:
Gelecekte asıl kritik beceri kod yazmak değil, davranış modellemektir. Çünkü düşük seviyeli kodlama giderek otomatikleşmektedir. Asıl mesele, bir robotun nasıl öğrenmesi gerektiğini, hangi hedefleri optimize edeceğini ve hangi bağlamda nasıl davranacağını tasarlamaktır. Bu, mühendislikten çok bilişsel sistem tasarımıdır.

1.13 Mini Uygulama

Bu bölümde teorik olarak ele alınan tüm kavramları küçük ama son derece öğretici bir problem üzerinden somutlaştırmak önemlidir. Çünkü robotik ve yapay zekâ gibi alanlarda gerçek anlayış, yalnızca tanımları bilmekle değil, bu tanımların basit bir sistem üzerinde nasıl davrandığını

gözlemleyebilmekle oluşur. Teori, ancak küçük bir dünyada çalıştığında zihinde gerçek bir karşılık bulur.

Şimdi oldukça basit görünen bir sistem düşünelim: beş çarpı beşlik bir grid (ızgara) ortamı. Bu ortamda hareket eden bir ajan vardır ve bu ajanın amacı belirli bir hedefe ulaşmaktır. Ajan yukarı, aşağı, sağ veya sol yönlerde hareket edebilir. Ortamda bazı engeller de bulunabilir. Hedefe ulaşmak, sistemin tek başarı kriteridir.

İlk bakışta bu problem son derece basit görünür. Hatta "bunun neresi yapay zekâ?" sorusu bile akla gelebilir. Ancak bu basitlik aldatıcıdır. Çünkü bu küçük problem, robotik ve yapay zekâ tasarımının iki temel yaklaşımını karşılaştırmak için ideal bir zihinsel laboratuvardır.

Şimdi temel soruyu soralım: bu ajanı nasıl tasarlarsınız?

Bu sorunun iki temel cevabı vardır. Birincisi, sistemi kurallarla yani açık if-else yapılarıyla tasarlamak; ikincisi ise sistemi öğrenme yoluyla tasarlamaktır. Bu iki yaklaşım, aslında modern akıllı sistemlerin iki farklı felsefesini temsil eder ve aralarındaki fark yalnızca teknik değil, derin bir zihniyet farkıdır.

İlk yaklaşımda ajan tamamen önceden tanımlanmış kurallarla çalışır. Yani sistemin davranışı insan tarafından açıkça yazılır. Örneğin hedef sağdaysa sağa git, soldaysa sola git, önünde engel varsa yön değiştir, hedefe ulaşıldığında dur gibi basit kurallar tanımlanır. Bu durumda ajan aslında karar veren bir varlık değil, verilen talimatları uygulayan bir mekanizmadır.

Bu yaklaşımın güçlü tarafı basitliktir. Sistem anlaşılırdır, kontrol edilebilirdir ve küçük problemler için hızlı çözümler üretir. Davranış tamamen tasarımcının elindedir. Ne yapılacağı bellidir, sürpriz yoktur.

Ancak bu netlik aynı zamanda büyük bir sınırlılık taşır. Çünkü sistem yalnızca önceden düşünülmüş durumlara tepki verebilir. Ortam değiştiğinde ya da yeni bir durum ortaya çıktığında sistem çöker ya da yeniden yazılması gerekir. Örneğin 5x5 grid 10x10 olduğunda ya da engeller rastgele dağıtıldığında, yazılan kurallar hızla yetersiz hale gelir. Çünkü if-else yapıları yalnızca bilinen dünyayı kapsar, bilinmeyenle baş edemez.

Bu noktada kritik bir gerçek ortaya çıkar: kurallarla çalışan sistemler yalnızca öngörülebilen dünyalar için uygundur. Oysa gerçek dünya öngörülebilir değildir. Karmaşıktır, dinamiktir ve sürekli değişir.

İkinci yaklaşım ise bu noktada devreye girer: öğrenme temelli sistemler. Bu yaklaşımda ajan için tek tek kurallar yazılmaz. Bunun yerine sistem, deneyim yoluyla nasıl davranacağını öğrenir. Başlangıçta rastgele hareket eden ajan, zaman içinde hangi davranışların hedefe götürdüğünü keşfeder.

Süreç genellikle oldukça sezgiseldir. Ajan hareket eder, hedefe ulaştığında ödül alır, yanlış hareket ettiğinde ceza alır. Bu geri bildirim döngüsü zamanla bir davranış modeline dönüşür. Sistem artık deneme yanılmadan öğrenmeye geçmiştir.

Bu yapı pekiştirmeli öğrenme fikrine dayanır. Burada önemli olan nokta şudur: davranış önceden yazılmaz, veriden ortaya çıkar. Sistem kendi stratejisini deneyim yoluyla inşa eder.

Öğrenen bir ajan yalnızca mevcut duruma değil, geçmiş deneyimlerine de dayanarak karar verir. Bu sayede daha önce hiç görmediği durumlara uyum sağlayabilir. Ortam değiştiğinde sistem tamamen çökmez; yeni durumu keşfetmeye başlar.

Örneğin 5x5 grid ortamı 10x10'a genişlediğinde, öğrenen bir sistem yeniden kurallara ihtiyaç duymaz. Çünkü davranış sabit değildir; öğrenme mekanizması vardır. Sistem yeni alanı keşfederek uyum sağlar.

Bu iki yaklaşımı yan yana koyduğumuzda fark çok netleşir. Kurallarla çalışan sistemler insan tarafından kontrol edilir, sabit davranış üretir ve değişime kapalıdır. Öğrenen sistemler ise veriden davranış üretir, dinamik ve adaptiftir, değişime açıktır ve olasılıksal bir yapıdadır.

Bu fark yalnızca teknik bir fark değildir. Bu fark, yapay zekâ ve robotik dünyasında bir zihniyet ayrımını temsil eder. Bir tarafta mühendislik mantığıyla yazılmış deterministik sistemler, diğer tarafta veriyle şekillenen adaptif sistemler vardır.

Şimdi daha temel bir soruya geliyoruz: hangisi daha esnektir?

Cevap nettir: öğrenme temelli yaklaşım daha esnektir. Çünkü kurallarla çalışan sistemler yalnızca bilinen durumlara göre tasarlanır. Her yeni durum için insan müdahalesi gerekir. Bu da sistemi kırılgan hale getirir. Öğrenen sistemler ise deneyimle gelişir ve zaman içinde kendini yeniden şekillendirebilir.

Esneklik burada kritik bir kavramdır. Çünkü gerçek dünya yalnızca bilinen problemlerden oluşmaz. Beklenmeyen durumlar, değişen koşullar ve yeni problemler sürekli ortaya çıkar. Bu nedenle sistemlerin bu değişime uyum sağlayabilmesi gerekir.

Bu küçük 5x5 grid problemi aslında çok daha büyük bir gerçeği temsil eder. Robotik ve yapay zekâ sistemlerinin temel problemi şudur: davranış nasıl oluşturulmalıdır?

Bu soruya iki cevap verilebilir: davranışı yazmak ya da davranışı öğrenmek. İlk yaklaşım klasik mühendisliktir. İkincisi ise modern yapay zekânın temelidir.

Bu nedenle bu küçük örnek, aslında büyük bir zihinsel dönüşümün modelidir. Sistem tasarlama biçimimiz değişmektedir.

Bu bölümün en önemli noktası teknik çözüm değil, zihinsel değişimdir. Bir sistem tasarlarken artık şu soru sorulmalıdır: hangi kuralları yazmalıyım değil, bu sistem nasıl öğrenebilir?

Bu soru değiştiğinde yaklaşım da değişir. Artık sistem sabit kurallarla değil, öğrenme mekanizmasıyla tasarlanır. Bu da robotik düşünmeye geçişi temsil eder.

Robotik düşünme şu üç temel özelliği içerir: sistem davranış üretir, çevreden öğrenir ve zaman içinde değişir. Bu nedenle sistemler artık statik değil, dinamik varlıklardır.

Sonuç olarak bu basit mini uygulama bize çok önemli bir şeyi öğretir. Sistemleri tasarlamanın iki yolu vardır: kurallar yazmak ya da öğrenme mekanizması kurmak. Kurallar basitlik sağlar ama esneklik vermez. Öğrenme karmaşıklık getirir ama esneklik kazandırır.

Modern dünyada sistemler giderek daha karmaşık hale geldiği için ikinci yaklaşım giderek daha önemli hale gelmektedir.

Ve bu bölümün temel mesajı şudur: geleceğin sistemleri kurallarla değil, öğrenmeyle tasarlanacaktır.

BÖLÜM 2 — ÖĞRENEN SİSTEMLERE GİRİŞ

Bu bölümde artık robotik sistemleri yalnızca kurallarla çalışan mekanizmalar olarak değil, veriden öğrenen ve zaman içinde davranışlarını değiştirebilen yapılar olarak ele almaya başlıyoruz. Birinci bölümde kurduğumuz temel ayrımlar—robotik, yapay zekâ ve programlama arasındaki farklar—şimdi daha ileri bir seviyeye taşınacaktır. Çünkü modern sistemlerin asıl gücü, önceden tanımlanmış kuralların değil, öğrenme kapasitesinin içinde gizlidir.

Öğrenen sistemler, çevreden gelen veriyi yalnızca işleyen değil, aynı zamanda bu veriden genelleme yapan yapılardır. Bu yaklaşımda sistemin davranışı sabit değildir; deneyimle birlikte şekillenir. Bu nedenle öğrenme, klasik mühendislikten farklı olarak dinamik bir tasarım süreci gerektirir. Artık "sistem ne yapmalı?" sorusundan çok, "sistem nasıl öğrenmeli?" sorusu önem kazanır.

Bu bölüm boyunca makine öğrenmesinin temel mantığını, veri–model ilişkisini ve öğrenmenin neden modern robotik sistemlerin merkezinde yer aldığını inceleyeceğiz. Ayrıca öğrenmenin yalnızca bir algoritma değil, aynı zamanda bir sistem davranışı üretme süreci olduğunu göreceğiz. Bu bakış açısı, ilerleyen bölümlerde PyTorch ve robotik uygulamalara geçiş için gerekli zihinsel zemini oluşturacaktır.

2.1. Veri Nedir, Öğrenme Nedir?

Öğrenen sistemleri anlamanın en temel adımı, iki kavramı netleştirmektir: veri ve öğrenme. Bu iki kavram çoğu zaman birlikte anılsa da, aslında farklı seviyelerde işleyen yapılardır. Veri ham bir malzemedir; öğrenme ise bu ham malzemenin anlamlı bir yapıya dönüştürülme sürecidir. Başka bir ifadeyle veri "dünya hakkında kayıtlar" ise, öğrenme bu kayıtların "dünya hakkında bir model" haline gelmesidir.

Modern yapay zekâ ve robotik sistemlerin merkezinde bu ilişki vardır. Bir sistemin akıllı olup olmadığını belirleyen şey, ne kadar veri topladığı değil, bu veriyi nasıl dönüştürdüğüdür. Çünkü veri tek başına hiçbir şey ifade etmez; anlam, ancak öğrenme süreciyle ortaya çıkar.

Veri Nedir?

Veri, en basit tanımıyla gözlem sonucu elde edilen sayısal ya da sembolik kayıtlardır. Bu kayıtlar, fiziksel dünyadan, dijital sistemlerden veya insan davranışlarından gelebilir. Bir sensörün ölçtüğü sıcaklık değeri, bir görüntüdeki piksel yoğunlukları, bir kullanıcının tıklama davranışı ya da bir robotun hareket konumu veri örnekleridir.

Ancak burada önemli bir nokta vardır: veri, anlam içermez. Veri yalnızca bir "durumun izidir". Örneğin 23°C sıcaklık değeri, tek başına ne iyi ne kötü ne de anlamlıdır. Ancak bağlam içine yerleştirildiğinde anlam kazanır.

Veriyi üç temel kategoriye ayırabiliriz:

1. Yapısal veri:
Tablolar, sayılar ve düzenli formatlarda bulunan veriler. Örneğin bir veritabanındaki müşteri kayıtları.
2. Yapısal olmayan veri:
Görüntüler, sesler, metinler gibi doğrudan tabloya dökülmeyen veriler. Örneğin bir fotoğraf.
3. Zaman serisi veri:
Zamana bağlı değişen veriler. Sensör okumaları, hisse senedi fiyatları veya robot hareketleri bu gruba girer.

Öğrenen sistemler için bu veri türlerinin hepsi kritiktir. Çünkü gerçek dünya sadece sayılardan değil, sürekli değişen ve çok boyutlu bilgi akışından oluşur.

Verinin Özelliği: Hamlık

Veriyi özel kılan şey onun "ham" olmasıdır. Ham veri işlenmemiştir, yorumlanmamıştır ve anlamlandırılmamıştır. Bu nedenle veri, tek başına bir karar veremez, bir sonuç üretemez.

Bir robot düşünelim. Bu robotun sensörleri çevresinden sürekli veri topluyor olsun. Bu veriler:

- Mesafe ölçümleri
- Görüntü piksel değerleri
- Motor pozisyonları

Bu bilgiler tek başına robotun ne yapacağını söylemez. Sadece "durum"u temsil eder. İşte bu noktada öğrenme devreye girer.

Öğrenme Nedir?

Öğrenme, veriden anlam çıkarma sürecidir. Bu süreçte bir sistem, geçmiş deneyimlere bakarak gelecekte nasıl davranması gerektiğini belirler. Yani öğrenme, veri ile davranış arasında bir köprü kurar.

Makine öğrenmesi açısından bakıldığında öğrenme şu şekilde tanımlanabilir:

👉 Bir sistemin, deneyim yoluyla performansını geliştirme süreci

Bu tanımda üç önemli unsur vardır:

- Deneyim (veri)
- Performans (başarı ölçütü)
- İyileştirme (optimizasyon)

Öğrenme süreci, bu üç unsurun etkileşimiyle oluşur.

Öğrenmenin Temel Mantığı

Bir öğrenme sistemi genellikle şu döngüyle çalışır:

1. Sistem veri alır
2. Bu veriye göre bir tahmin yapar
3. Tahminin doğruluğu ölçülür
4. Hata hesaplanır
5. Sistem kendini günceller

Bu döngü tekrarlandıkça sistem daha iyi hale gelir.

Buradaki en önemli kavram "hata"dır. Çünkü öğrenme, hatasızlıktan değil, hatadan doğar. Sistem yanlış yaptıkça öğrenir ve bu yanlışları azaltmaya çalışır.

Veri ile Öğrenme Arasındaki İlişki

Veri ve öğrenme arasındaki ilişki tek yönlü değildir. Veri olmadan öğrenme gerçekleşemez, ancak öğrenme olmadan veri anlamsızdır.

Bu ilişkiyi şöyle düşünebiliriz:

- Veri = deneyim
- Öğrenme = deneyimden anlam çıkarma

Bir insan nasıl deneyimlerinden öğreniyorsa, yapay sistemler de veriden öğrenir. Ancak önemli bir fark vardır: insan öğrenmesi biyolojik ve sezgiseldir, makine öğrenmesi ise matematiksel ve hesaplamalıdır.

Öğrenme Neden Gereklidir?

Geleneksel sistemler kurallara dayanır. Bu sistemlerde her durum için bir kural yazılır. Ancak gerçek dünya son derece karmaşıktır ve tüm durumları önceden yazmak mümkün değildir.

Örneğin:

- Bir robot her ışık koşulunu önceden bilemez
- Bir otonom araç tüm trafik senaryolarını önceden tanımlayamaz

- Bir görüntü tanıma sistemi tüm nesneleri tek tek kodlayamaz

İşte bu nedenle öğrenme gerekir.

Öğrenme sayesinde sistem:

- Yeni durumlara uyum sağlar
- Daha önce görmediği örnekleri genelleyebilir
- Zamanla performansını artırabilir

Öğrenme Türleri

Makine öğrenmesinde üç temel öğrenme türü vardır:

1. Denetimli öğrenme:
Sistem, giriş ve doğru çıkış örnekleriyle eğitilir. Amaç, doğru eşlemeyi öğrenmektir.

2. Denetimsiz öğrenme:
Sisteme sadece veri verilir, doğru cevap verilmez. Sistem verideki yapıyı kendisi keşfeder.

3. Pekiştirmeli öğrenme:
Sistem, ödül ve ceza mekanizmasıyla öğrenir. Doğru davranışlar ödüllendirilir, yanlışlar cezalandırılır.

Bu üç yaklaşım, modern yapay zekânın temelini oluşturur.

Veri Kalitesi ve Öğrenme

Öğrenmenin başarısı büyük ölçüde verinin kalitesine bağlıdır. Çünkü kötü veri, iyi öğrenme üretemez.

Bu nedenle veri:

- Doğru olmalı
- Temsil gücü yüksek olmalı
- Yeterli çeşitlilik içermeli

Aksi durumda sistem yanlış öğrenir. Bu durum “çöp girerse çöp çıkar” prensibiyle özetlenir.

Öğrenme Bir Davranış Üretme Sürecidir

Öğrenme sadece bir analiz süreci değildir. Aynı zamanda bir davranış üretme sürecidir. Çünkü öğrenen sistemler yalnızca veriyi anlamaz, aynı zamanda bu anlayışa göre hareket eder.

Örneğin bir robot:

- Engel gördüğünde durmayı öğrenebilir
- Belirli bir nesneyi takip etmeyi öğrenebilir

- Hedefe ulaşma stratejisi geliştirebilir

Bu davranışlar, doğrudan veriden türetilir.

Veri Olmadan Öğrenme Mümkün mü?

Hayır. Veri olmadan öğrenme mümkün değildir. Çünkü öğrenme, deneyime dayanır. Deneyim ise veridir. Ancak önemli bir nüans vardır: sadece veri de yeterli değildir. Veri, doğru bir model ve doğru bir öğrenme algoritması ile birleşmelidir.

Sonuç: Veri ve Öğrenmenin Birliği

Bu bölümde temel iki kavramı derinlemesine ele aldık: veri ve öğrenme. Artık net bir şekilde görüyoruz ki bu iki kavram birbirinden ayrı düşünülemez.

- Veri ham bilgidir
- Öğrenme bu bilgiyi anlamlı hale getirir
- Birlikte akıllı sistemleri oluştururlar

Modern robotik ve yapay zekâ sistemlerinin temelinde bu ilişki vardır. Veri olmadan öğrenme kördür, öğrenme olmadan veri anlamsızdır.

Bu nedenle öğrenen sistemleri anlamak için önce veriyi, sonra öğrenmeyi doğru anlamak gerekir. Bu kitap boyunca ilerledikçe bu iki kavramın nasıl birleşerek davranış üreten sistemlere dönüştüğünü adım adım göreceğiz.

2.2. Fonksiyon Öğrenmek: Sezgisel Bir Yaklaşım

Fonksiyon öğrenme kavramı, modern yapay zekâ ve makine öğrenmesinin en temel ama aynı zamanda en sezgisel yapı taşlarından biridir. Ancak bu kavram çoğu zaman matematiksel formüller içinde kaybolur ve özündeki basit fikir görünmez hale gelir. Oysa fonksiyon öğrenmeyi anlamak için karmaşık denklemlerden başlamaya gerek yoktur. Asıl mesele, bir sistemin “girdi” ile “çıktı” arasındaki ilişkiyi nasıl keşfettiğini anlamaktır.

En basit haliyle bir fonksiyon, bir girdiyi alıp bir çıktıya dönüştüren bir mekanizmadır. Matematiksel olarak bu şu şekilde ifade edilir:

$y = f(x)$

Burada x giriş, y ise çıktıdır. Ancak makine öğrenmesi açısından bakıldığında bu ifade çok daha derin bir anlama sahiptir. Çünkü burada önemli olan şey fonksiyonun kendisi değil, bu fonksiyonun **veriden öğrenilmesidir**.

Fonksiyon Öğrenme Nedir?

Fonksiyon öğrenme, bir sistemin örnek verilerden yola çıkarak giriş–çıkış ilişkisini keşfetme sürecidir. Başka bir ifadeyle sistem, doğru fonksiyonun ne olduğunu önceden bilmez; onu veriden tahmin eder.

Örneğin elimizde şu tür bir veri olsun:

- 1 → 2
- 2 → 4
- 3 → 6

Burada sistemin öğrenmesi gereken şey şudur: girişin iki katı çıkıştır. Yani fonksiyon f(x) = 2x'tir.

Ancak makine öğrenmesinde bu ilişki önceden verilmez. Sistem yalnızca örnekleri görür ve aralarındaki yapıyı kendisi keşfetmeye çalışır.

Bu süreçte temel soru şudur:

👉 "Bu veriyi üreten fonksiyon nedir?"

İnsan Sezgisi ile Makine Öğrenmesi Arasındaki Bağ

İnsanlar aslında sürekli fonksiyon öğrenir. Bir çocuk dünyayı gözlemlerken şunu yapar:

- Bir nesneyi iter → hareket eder
- Daha güçlü iter → daha hızlı hareket eder
- Hiç itmez → hareket etmez

Çocuk bu ilişkiden şu sonucu çıkarır: "kuvvet arttıkça hareket artar".

Bu bir fonksiyon öğrenme sürecidir, ancak bilinçsiz ve sezgiseldir. Makine öğrenmesi ise aynı süreci matematiksel olarak formalize eder. Yani sezgiyi sayısal modele dönüştürür.

Fonksiyon Öğrenmenin Temel Problemi

Fonksiyon öğrenmenin temel problemi şudur:

👉 Gerçek fonksiyon bilinmez

Elimizde sadece örnek veriler vardır. Bu veriler gürültülü, eksik veya hatalı olabilir. Sistem bu verilerden "en iyi tahmini" yapmak zorundadır.

Bu nedenle makine öğrenmesi süreci bir tahmin problemidir:

- Gerçek fonksiyon: bilinmiyor
- Veri: sınırlı
- Amaç: en iyi yaklaşımı bulmak

Şekil 3. Fonksiyon Öğrenmenin Temel Problemi

Model Nedir?

Fonksiyon öğrenme sürecinde "model" kavramı ortaya çıkar. Model, fonksiyonun yaklaşık bir temsilidir.

Başka bir ifadeyle:

👉 Model = öğrenilmeye çalışan fonksiyonun tahmini

Modelin amacı, verilen giriş için en doğru çıktıyı üretmektir.

Örneğin:

- Girdi: evin büyüklüğü
- Çıktı: ev fiyatı

Sistem bu ilişkiyi öğrenmeye çalışır. Ancak gerçek dünya ilişkileri hiçbir zaman mükemmel değildir. Bu nedenle model, yaklaşık bir fonksiyon öğrenir.

Hata Kavramı: Öğrenmenin Motoru

Fonksiyon öğrenme sürecinin kalbinde "hata" vardır. Hata, modelin tahmini ile gerçek değer arasındaki farktır.

Eğer model şu tahmini yapıyorsa:

- Gerçek: 10
- Tahmin: 8

Hata = 2

Bu hata, öğrenme sürecinin yönünü belirler. Sistem bu hatayı azaltmaya çalışır.

Öğrenme süreci şu döngüyle ilerler:

1. Tahmin yapılır
2. Hata hesaplanır
3. Model güncellenir
4. Hata azaltılır

Bu döngü tekrarlandıkça model gerçek fonksiyona yaklaşır.

Fonksiyon Öğrenme = Yaklaşım Süreci

Fonksiyon öğrenme sürecini en iyi şu metafor açıklar:

👉 Gerçek fonksiyon bir dağdır, model ise o dağa tırmanmaya çalışan bir kişidir.

Başlangıçta model çok uzaktadır. Ancak her adımda biraz daha doğru yönü öğrenir ve hedefe yaklaşır. Burada önemli olan nokta şudur:

- Model hiçbir zaman mükemmel olmayabilir
- Ama giderek daha iyi hale gelir

Bu, öğrenmenin temel doğasıdır.

Neden Fonksiyon Öğreniriz?

Gerçek dünya problemleri doğrudan formülle açıklanamaz. Çünkü:

- İlişkiler karmaşıktır
- Değişken sayısı fazladır
- Gürültü vardır
- Belirsizlik yüksektir

Bu nedenle klasik matematiksel modeller yetersiz kalır.

Fonksiyon öğrenme bize şunu sağlar:

👉 Karmaşık ilişkileri veriden otomatik olarak keşfetmek

Bu sayede:

- Görüntüler sınıflandırılabilir
- Sesler tanınabilir
- Robotlar kontrol edilebilir
- Tahmin sistemleri kurulabilir

Lineer ve Lineer Olmayan Fonksiyonlar

Fonksiyon öğrenme sürecinde iki temel yapı vardır:

Lineer fonksiyonlar:
Basit doğrusal ilişkiler içerir. Örneğin f(x) = ax + b

Lineer olmayan fonksiyonlar:
Daha karmaşık, eğrisel ve çok katmanlı ilişkiler içerir. Gerçek dünya problemleri çoğunlukla lineer değildir. Bu nedenle modern yapay zekâ sistemleri lineer olmayan fonksiyonları öğrenmeye çalışır.

Öğrenme Neden Zordur?

Fonksiyon öğrenme kolay bir süreç değildir. Çünkü:

- Veri eksik olabilir
- Gürültü olabilir
- İlişki çok karmaşık olabilir
- Aynı giriş farklı çıktılar üretebilir

Bu nedenle sistem hiçbir zaman "kesin doğru"yu bilmez. Sadece en iyi tahmini yapar.

Genelleme: Öğrenmenin Gerçek Hedefi

Fonksiyon öğrenmede en önemli kavramlardan biri genellemedir.

Genelleme şudur:

👉 Görülmeyen veriler için doğru tahmin yapabilme yeteneği

Bir model sadece gördüğü veriyi ezberlerse başarısız olur. Gerçek başarı, hiç görmediği örneklerde doğru çalışabilmesidir.

Ezberleme vs Öğrenme

Fonksiyon öğrenme sürecinde iki durum vardır:

- Ezberleme (memorization)
- Öğrenme (generalization)

Ezberleme, sadece görülen veriyi tekrar etmektir. Öğrenme ise verinin arkasındaki yapıyı keşfetmektir.

Örneğin:

- Ezberleyen model: 1→2, 2→4, 3→6 dışında hiçbir şey bilemez
- Öğrenen model: "x'in iki katı" ilişkisini keşfeder

Fonksiyon Öğrenmenin Robotik ile İlişkisi

Robotik sistemlerde fonksiyon öğrenme kritik bir rol oynar. Çünkü robotlar:

- Sensör verisi alır
- Bu veriyi yorumlar
- Hareket üretir

Bu süreç aslında bir fonksiyon öğrenme problemidir:

👉 Sensör verisi → davranış

Örneğin bir robot:

- Kamera görüntüsünden nesne konumunu öğrenir
- Mesafe verisinden hareket stratejisi üretir

Sezgisel Sonuç

Fonksiyon öğrenmeyi anlamanın en basit yolu şudur:

👉 Dünya = giriş
👉 Davranış = çıkış
👉 Öğrenme = bu ilişkiyi keşfetme süreci

Bu bakış açısı, tüm yapay zekâ sistemlerinin temelidir.

Sonuç

Fonksiyon öğrenmek, sadece matematiksel bir işlem değil, bir sistemin dünyayı anlamaya çalışmasıdır. Veri ile başlayan bu süreç, model ile şekillenir ve öğrenme ile gelişir.

Bu bölümün temel mesajı şudur:

- Fonksiyonlar önceden verilmez
- Fonksiyonlar veriden öğrenilir
- Öğrenme, yaklaşım sürecidir

Ve bu yaklaşım, modern yapay zekânın kalbini oluşturur.

2.3. Vektörler ve Matrislerle Düşünmek

Modern yapay zekâ ve robotik sistemleri anlamanın en kritik adımlarından biri, dünyayı artık tekil sayılarla değil, çok boyutlu yapılarla düşünmeye başlamaktır. Bu dönüşüm ilk bakışta teknik bir ayrıntı gibi görünür, ancak aslında zihinsel bir kırılmadır. Çünkü vektörler ve matrisler yalnızca matematiksel araçlar değildir; aynı zamanda bilgiyi temsil etme biçimidir.

Bir sistemin “akıllı” olabilmesi için, dünyayı tek bir değişken üzerinden değil, çok sayıda değişkenin birlikte oluşturduğu bir yapı olarak görmesi gerekir. İşte bu noktada vektörler

devreye girer. Vektörler, bilgiyi birden fazla bileşenle temsil etmenin en doğal yoludur. Matrisler ise bu vektörlerin birlikte nasıl işlendiğini tanımlar.

Bu bölümde amaç, vektör ve matrisleri soyut matematik kavramları olarak değil, düşünme araçları olarak anlamaktır.

Skaler Düşünmeden Vektörel Düşünmeye Geçiş

Geleneksel düşünme biçiminde çoğu zaman tek bir değer üzerinden karar veririz. Örneğin:

- Sıcaklık: 25°C
- Hız: 60 km/s
- Fiyat: 100 TL

Bu tür tekil değerler "skaler" olarak adlandırılır. Ancak gerçek dünya nadiren tek bir değişkenle açıklanabilir. Bir sistemin durumu genellikle birçok değişkenin birleşimidir.

Örneğin bir robotu düşünelim:

- X konumu
- Y konumu
- Hız
- Yön
- Sensör verileri

Bu bilgilerin her biri tek başına yeterli değildir. Ancak birlikte bir "durum" oluştururlar.

İşte bu durum vektör ile temsil edilir.

👉 Vektör = bir sistemin çok boyutlu temsili

Vektör Nedir?

Vektör, birden fazla sayının belirli bir düzen içinde bir araya gelmesidir. Matematiksel olarak şöyle yazılabilir:

$(x_1, x_2, x_3, ..., x_n)$

Bu yapı, bir sistemin durumunu temsil eder. Yapay zekâ sistemlerinde vektörler şu amaçlarla kullanılır:

- Görüntüleri temsil etmek
- Metinleri sayısallaştırmak
- Sensör verilerini modellemek

- Robot durumlarını tanımlamak

Örneğin bir görüntü, milyonlarca pikselden oluşur. Bu pikseller bir vektör haline getirildiğinde sistem tarafından işlenebilir hale gelir.

Vektörler Neden Önemlidir?

Vektörlerin önemi, bilgiyi "hesaplanabilir" hale getirmesinden gelir. Bilgisayarlar tek tek anlamlar yerine sayısal yapılarla çalışır. Bu nedenle dünya, vektörlere dönüştürülür.

Vektörler sayesinde:

- Benzerlik ölçülebilir
- Mesafeler hesaplanabilir
- Örüntüler bulunabilir
- Tahmin yapılabilir

Örneğin iki görüntü vektörü arasındaki mesafe küçükse, bu görüntüler benzer kabul edilir.

Matris Nedir?

Eğer vektör bir liste ise, matris bu listelerin bir araya gelmiş halidir. Başka bir ifadeyle matris, verinin iki boyutlu temsilidir.

Matris şöyle yazılır:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Matrisler, özellikle çok sayıda verinin birlikte işlendiği durumlarda kullanılır.

Matrisler Ne İşe Yarar?

Matrislerin en önemli işlevi, dönüşümleri temsil etmektir. Bir vektör bir matrisle çarpıldığında, o vektör başka bir forma dönüşür.

Bu şu anlama gelir:

👉 Matris = dönüşüm operatörü

Örneğin bir görüntü işleme sisteminde:

- Girdi görüntü
- Matris = filtre
- Çıktı = dönüştürülmüş görüntü

Bu yapı sayesinde sistemler karmaşık işlemleri hızlıca gerçekleştirebilir.

Vektör ve Matrislerin Robotik Anlamı

Robotik sistemleri anlamanın en kritik aşamalarından biri, fiziksel dünyanın nasıl matematiksel bir dile çevrildiğini kavramaktır. Bu dönüşümün merkezinde ise vektörler ve matrisler yer alır. Çünkü robotlar, insan gibi sezgisel bir şekilde "uzayı" algılamaz; bunun yerine konumları, hızları, yönleri ve çevresel bilgileri sayısal yapılara dönüştürerek işler. Bu sayısal yapıların en temel temsil biçimi vektörlerdir.

Bir robotun dünyadaki varlığını tanımlamak için yalnızca "nerede olduğu" bilgisi yeterli değildir. Aynı zamanda nasıl hareket ettiği, hangi yönde ilerlediği ve çevresinin ona nasıl etki ettiği de bilinmelidir. Örneğin bir mobil robotu düşünelim. Bu robotun konumu, iki veya üç boyutlu bir koordinat sistemi içinde bir vektör olarak ifade edilir. Bu vektör, robotun fiziksel uzaydaki yerini belirler. Ancak bu bilgi tek başına statik bir bilgidir. Robotun gerçekten "yaşayan" bir sistem gibi davranabilmesi için bu konumun zaman içinde nasıl değiştiği de bilinmelidir.

Bu noktada hız ve ivme kavramları devreye girer. Hız vektörü, robotun konumunun zamana göre değişimini ifade eder. İvme vektörü ise hızın değişimini temsil eder. Bu üç kavram bir araya geldiğinde robotun fiziksel davranışını tanımlayan temel dinamik yapı ortaya çıkar. Ancak modern robotik sistemlerde yalnızca bu üçlü yeterli değildir. Sensörlerden gelen bilgiler de bu yapıya eklenmelidir. Örneğin bir robotun önündeki mesafeyi ölçen sensör, çevresel bir bilgi üretir ve bu bilgi de ayrı bir vektör olarak temsil edilir.

Bu farklı vektörler bir araya geldiğinde "durum vektörü" adı verilen daha büyük bir yapı oluşur. Durum vektörü, robotun o anki tüm fiziksel ve algısal bilgisini tek bir matematiksel yapı içinde toplar. Bu yapı sayesinde robot, kendi varlığını sayısal bir bütün olarak temsil edebilir. Başka bir ifadeyle, robotun "kim olduğu" ve "ne yaptığı" bilgisi bu vektör içinde saklanır.

Ancak vektörler yalnızca durumu temsil eder; hareketi ve dönüşümü açıklamak için matrislere ihtiyaç vardır. Robotik sistemlerde matrisler, bir durumdan başka bir duruma geçişi tanımlar. Örneğin bir robotun yönünü değiştirmesi, matematiksel olarak bir dönüşüm işlemi ile ifade edilir. Bu dönüşüm genellikle bir dönüş matrisi ile temsil edilir. Dönüş matrisi, robotun uzaydaki yönünü değiştirirken, konumunu belirli bir kurala göre yeniden hesaplar. Bu sayede robotun fiziksel hareketi matematiksel olarak modellenmiş olur.

Benzer şekilde kontrol matrisleri, robotun motorlarına gönderilen sinyallerin nasıl fiziksel harekete dönüşeceğini belirler. Bir robotun motoru doğrudan "ileri git" komutunu anlamaz; bunun yerine sayısal bir sinyal alır ve bu sinyali mekanik harekete çevirir. Kontrol matrisi, bu dönüşümün matematiksel modelidir. Sensör matrisleri ise dış dünyadan gelen verilerin nasıl işleneceğini tanımlar. Örneğin bir kamera görüntüsü, çok boyutlu bir veri yapısıdır ve bu veri doğrudan değil, matris dönüşümleri üzerinden anlamlandırılır.

Bu yapıların en önemli özelliği, robotik sistemlerin fiziksel dünyayı tamamen sayısal işlemlerle temsil edebilmesini sağlamasıdır. Eğer vektörler ve matrisler olmasaydı, bir robotun hareketini hesaplamak, çevresini analiz etmek veya gelecekteki durumunu tahmin etmek mümkün

olmazdı. Çünkü fiziksel dünya sürekli ve karmaşıktır; matrisler ise bu sürekliliği hesaplanabilir bir yapıya dönüştürür.

Bu nedenle modern robotik sistemlerde vektörler ve matrisler yalnızca matematiksel araçlar değil, aynı zamanda düşünme biçiminin temelidir. Bir robotun "düşünmesi", aslında bu vektör ve matrislerin sürekli olarak güncellenmesi ve dönüştürülmesi anlamına gelir. Konum değişir, hız güncellenir, sensör verisi yeniden hesaplanır ve kontrol matrisleri yeni kararlar üretir. Bu döngü sürekli tekrar eder ve robotun davranışını oluşturur.

Sonuç olarak, robotik sistemlerde vektörler ve matrisler fiziksel dünyanın dijital bir yansımasıdır. Bu yapı olmadan robotların hareketini anlamak, kontrol etmek veya geliştirmek mümkün değildir. Çünkü robotik, özünde fiziksel dünyanın matematiksel bir dile çevrilmesidir ve bu dilin temel harfleri vektörler, temel grameri ise matrislerdir.

Neden Vektörlerle Düşünmeliyiz?

Vektörlerle düşünmek, dünyayı çok boyutlu görmek demektir. Bu bakış açısı, özellikle yapay zekâ sistemlerinde kritik öneme sahiptir.

Çünkü gerçek dünya:

- Tek boyutlu değildir
- Sabit değildir
- Sürekli değişir

Vektörler bu değişimi temsil etmenin en doğal yoludur.

Örnek: Bir Görüntü Nasıl Vektöre Dönüşür?

Bir görüntü düşünelim. Bu görüntü:

- 1000 x 1000 piksel olabilir

Her piksel bir sayıdır. Bu durumda görüntü:

👉 1 milyon boyutlu bir vektör haline gelir

Bu vektör artık matematiksel olarak işlenebilir hale gelir.

Vektör Uzayında Düşünmek

Vektörler sadece veri değil, aynı zamanda bir "uzay" oluşturur. Bu uzaya vektör uzayı denir.

Bu uzayda:

- Benzer noktalar birbirine yakındır
- Farklı noktalar uzaktır

Bu yapı sayesinde:

- Kümeleme yapılabilir
- Sınıflandırma yapılabilir
- Örüntüler bulunabilir

Matris Çarpımı Ne Anlama Gelir?

Matris çarpımı, bir sistemin bir başka sisteme dönüşmesidir. Bu dönüşüm:

- Veri değişimi
- Özellik çıkarımı
- Boyut azaltma

gibi işlemleri temsil eder.

Örneğin bir yapay sinir ağı katmanı aslında bir matristir. Girdi vektörü bu matrisle çarpıldığında yeni bir temsil elde edilir.

Öğrenme ile Vektörler Arasındaki Bağ

Makine öğrenmesinde her şey vektörlerle başlar. Model:

- Girdiyi vektöre çevirir
- Bu vektör üzerinde işlem yapar
- Yeni bir vektör üretir

Bu süreç öğrenmenin temelidir.

Öğrenme aslında şudur:

👉 Vektörleri doğru şekilde dönüştürmeyi öğrenmek

Sezgisel Bir Bakış

Vektörleri anlamanın en kolay yolu şudur:

- Tek sayı = nokta
- Vektör = yönü olan bilgi
- Matris = dönüşüm mekanizması

Bu bakış açısı, karmaşık matematiksel işlemleri sezgisel hale getirir.

Neden Bu Kadar Önemli?

Vektör ve matrisler olmadan modern yapay zekâ mümkün değildir. Çünkü:

- Veri çok boyutludur

- İlişkiler karmaşıktır
- Hesaplamalar büyük ölçeklidir

Bu nedenle tüm modern modeller:

- Sinir ağları
- Görüntü işleme sistemleri
- Dil modelleri

vektör ve matrisler üzerine kuruludur.

Sonuç

Vektörler ve matrisler, sadece matematiksel araçlar değildir. Onlar, modern sistemlerin dünyayı nasıl gördüğünü belirleyen temel yapılardır.

Bu bölümün ana fikri şudur:

- Vektör = bilgiyi temsil etme biçimi
- Matris = bilgiyi dönüştürme biçimi
- Öğrenme = bu dönüşümleri optimize etme süreci

Bu bakış açısını kazandığınızda, artık dünyayı tekil sayılarla değil, çok boyutlu yapılarla düşünmeye başlarsınız. Bu da sizi klasik programlamadan çıkarıp modern yapay zekâ düşüncesine yaklaştırır.

2.4. PyTorch ile İlk Model

Öğrenen sistemleri anlamanın en kritik aşamalarından biri, teorik kavramları somut bir modele dönüştürebilmektir. Çünkü yapay zekâ yalnızca fikirlerden oluşmaz; bu fikirler bir yazılım altyapısı üzerinde çalıştığında gerçek anlam kazanır. Bu noktada PyTorch, modern makine öğrenmesinin en önemli araçlarından biri olarak karşımıza çıkar.

Ancak PyTorch'u yalnızca bir "kütüphane" olarak görmek eksik bir bakış olur. PyTorch, aslında öğrenen sistemleri inşa etmenin bir düşünme modelidir. Bu model, veriyi nasıl temsil edeceğimizi, modeli nasıl kuracağımızı ve öğrenme sürecini nasıl yöneteceğimizi belirler.

Bu bölümde amaç, ilk kez bir öğrenen model kurmak ve bu model üzerinden öğrenmenin nasıl gerçekleştiğini sezgisel olarak anlamaktır.

Neden Bir Model Kuruyoruz?

Yapay zekâ alanında en temel soru şudur:

👉 Bir sistem veriden nasıl öğrenir?

Bu sorunun cevabı soyut düzeyde anlaşılabilir, ancak gerçek anlamda kavranması için bir model kurulması gerekir. Çünkü öğrenme, ancak bir model üzerinde gözlemlendiğinde anlaşılır hale gelir.

Bir model kurmak şunları sağlar:

- Veriyi anlamlandırma
- Tahmin üretme
- Hata hesaplama
- Modeli güncelleme

Bu dört adım, öğrenmenin döngüsünü oluşturur.

PyTorch'un Temel Felsefesi

PyTorch'un en önemli özelliği, hesaplamaları dinamik bir şekilde yapmasıdır. Bu ne demektir?

Klasik programlama dillerinde işlemler sabittir. Ancak PyTorch'ta model, çalıştıkça şekillenir. Bu nedenle PyTorch, "dinamik hesaplama grafiği" kullanır.

Bu yaklaşımın anlamı şudur:

👉 Model sadece yazılmaz, aynı zamanda çalışırken inşa edilir.

Bu özellik, öğrenen sistemler için çok önemlidir çünkü öğrenme süreci statik değildir; sürekli değişir.

İlk Model: En Basit Öğrenme Problemi

Şimdi çok basit bir problem düşünelim:

- Girdi: x
- Çıktı: y

Ve aralarında basit bir ilişki olsun:

👉 y = 2x

Bu ilişkiyi sisteme öğretmek istiyoruz. Ancak biz bu kuralı sisteme vermiyoruz. Sistem bunu veriden öğrenmek zorunda.

Elimizde şu örnekler olsun:

- 1 → 2
- 2 → 4
- 3 → 6

- 4 → 8

Amaç: sistemin bu ilişkiyi keşfetmesi.

Model Nedir?

Model, girdiyi alıp çıktı üreten bir fonksiyondur. Ancak burada önemli olan nokta şudur:

👉 Model önceden doğru değildir, öğrenerek doğru hale gelir.

Bu nedenle model başlangıçta rastgele davranır. Zamanla hata yapar, bu hatalardan öğrenir ve kendini düzeltir.

PyTorch'ta Model Kurma Mantığı

PyTorch'ta bir model genellikle şu bileşenlerden oluşur:

- Katmanlar (layers)
- Ağırlıklar (weights)
- Aktivasyon fonksiyonları
- İleri yayılım (forward pass)
- Geri yayılım (backpropagation)

Bu bileşenler birlikte çalışarak öğrenme sürecini oluşturur.

Öğrenmenin Kalbi: Hata

Modelin öğrenmesi için bir ölçüte ihtiyacı vardır. Bu ölçüt "hata"dır.

Hata şu şekilde tanımlanır:

👉 Gerçek değer – Tahmin edilen değer

Örneğin:

- Gerçek: 4
- Tahmin: 3
- Hata: 1

Model bu hatayı azaltmaya çalışır.

Öğrenme Döngüsü

PyTorch ile bir modelin öğrenme süreci şu döngüye dayanır:

1. Girdi modele verilir
2. Model tahmin yapar

3. Hata hesaplanır
4. Ağırlıklar güncellenir
5. Süreç tekrar edilir

Bu döngü binlerce kez tekrarlandığında model öğrenmiş olur.

Ağırlıklar Nedir?

Ağırlıklar, modelin "öğrendiği bilgi"yi temsil eder. Başlangıçta rastgele olan bu değerler, öğrenme süreciyle birlikte optimize edilir.

Ağırlıkların görevi şudur:

👉 Girdiyi doğru çıktıya dönüştürmek

Bu nedenle öğrenme aslında ağırlıkların doğru değerleri bulma sürecidir.

Backpropagation: Öğrenmenin Mekanizması

Backpropagation (geri yayılım), hatanın modele geri gönderilerek ağırlıkların güncellenmesini sağlayan mekanizmadır.

Bu süreç şu şekilde çalışır:

- Model hata yapar
- Hata hesaplanır
- Hata geriye doğru yayılır
- Ağırlıklar güncellenir

Bu süreç olmadan öğrenme mümkün değildir.

Sezgisel Anlam

Bu süreci sezgisel olarak şöyle düşünebiliriz:

- Model bir tahmin yapar
- Yanlış yaptığını görür
- Kendini düzeltir
- Bir dahaki sefere daha iyi yapar

Bu, insan öğrenmesine oldukça benzer.

İlk Modelin Önemi

Bu basit örnek aslında çok önemli bir gerçeği gösterir:

👉 Karmaşık yapay zekâ sistemleri, basit öğrenme döngülerinden oluşur.

Bugün kullandığımız büyük dil modelleri veya görüntü tanıma sistemleri bile aynı temel prensiplere dayanır.

Neden PyTorch?

PyTorch'un tercih edilme nedeni, öğrenme sürecini şeffaf hale getirmesidir. Kullanıcı:

- Veriyi görebilir
- Modeli anlayabilir
- Öğrenme sürecini izleyebilir

Bu sayede sistem "kara kutu" olmaktan çıkar ve anlaşılabilir hale gelir.

Model = Fonksiyon Öğrenme Makinesi

Bu noktada önemli bir bağlantı kurabiliriz:

👉 Model = fonksiyon öğrenen sistem

Bir model aslında şu problemi çözer:

- Bilinmeyen fonksiyonu veriden bulmak

Örneğin:

- x → y ilişkisini öğrenmek
- Görüntü → etiket ilişkisini öğrenmek
- Sensör → hareket ilişkisini öğrenmek

Öğrenme Neden Çalışır?

Öğrenme sistemlerinin çalışmasının temel nedeni şudur:

👉 Dünya düzenlidir ama bu düzen gizlidir

Model bu gizli düzeni veriden çıkarır.

Hata Azaltma = Öğrenme

Öğrenme süreci aslında bir optimizasyon problemidir. Amaç:

👉 Hatayı minimize etmek

Bu nedenle model sürekli kendini geliştirir.

Basit Model, Büyük Fikir

Bu bölümde kurulan model çok basittir. Ancak temsil ettiği fikir çok büyüktür:

- Sistemler öğrenebilir
- Davranış veriden doğabilir
- Kurallar yerine veri kullanılabilir

Bu fikir modern yapay zekânın temelidir.

Sonuç

PyTorch ile ilk modelimizi kurarak çok önemli bir dönüşüm gerçekleştirdik. Artık sistemleri yalnızca tanımlamıyoruz; onların öğrenmesini gözlemliyoruz.

Bu bölümün temel mesajı şudur:

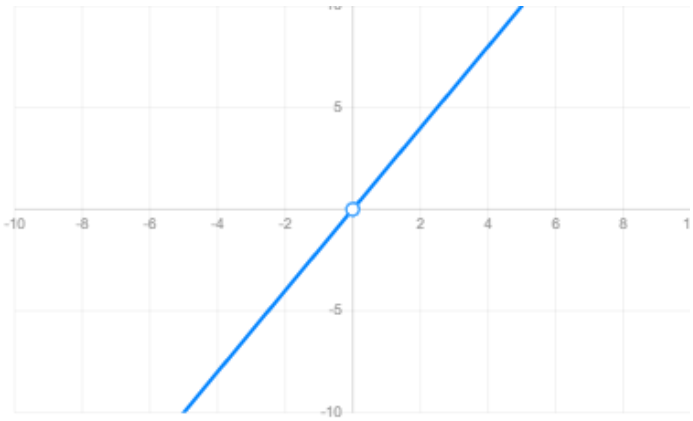
- Model = öğrenen fonksiyon
- Veri = deneyim
- Hata = öğrenmenin motoru
- PyTorch = bu sürecin aracı

Bu noktadan sonra artık yalnızca programlama değil, öğrenen sistem tasarımı yapmaya başlıyoruz.

2.5. Basit Bir PyTorch Örneği

Şimdiye kadar makine öğrenmesinin temel mantığını kavramsal olarak ele aldık. Modelin bir fonksiyon öğrenme sistemi olduğunu, verilerden örüntüler keşfettiğini ve hata yaparak öğrendiğini gördük. Ancak bütün bu fikirler hâlâ soyut görünebilir. Çünkü öğrenmenin gerçekten nasıl gerçekleştiğini anlamanın en iyi yolu, çalışan küçük bir sistemi gözlemlemektir.

Şekil 4. y=2x Fonksiyonu



Bu nedenle şimdi PyTorch kullanarak son derece basit bir model kuracağız. Amacımız karmaşık bir yapay zekâ sistemi geliştirmek değildir. Buradaki amaç, öğrenmenin temel mekanizmasını görünür hale getirmektir.

Kuracağımız sistem yalnızca çok basit bir ilişkiyi öğrenecek:

İnsan için bu ilişki son derece açıktır. Eğer x = 3 ise y = 6 olur. Eğer x = 10 ise y = 20 olur. Biz bu kuralı birkaç saniyede anlayabiliriz. Ancak makine için durum farklıdır. Bilgisayar bu ilişkiyi "kendiliğinden" bilmez. Ona yalnızca örnekler verilir.

Örneğin:

- 1 → 2
- 2 → 4
- 3 → 6
- 4 → 8

Sistem bu örneklerden yola çıkarak aradaki gizli ilişkiyi keşfetmeye çalışacaktır.

Buradaki önemli nokta şudur:

Biz sisteme kuralı vermiyoruz. Sistem kuralı veriden öğreniyor. Makine öğrenmesinin özü tam olarak budur.

İlk Adım: Veriyi Tanımlamak

Öncelikle sisteme örnek veriler vermemiz gerekir. PyTorch'ta veriler genellikle ensör adı verilen veri yapıları içinde tutulur. Tensor kavramı ilk başta karmaşık görünebilir, ancak sezgisel olarak onları "çok boyutlu sayı tabloları" gibi düşünebiliriz. Kodumuz şu şekilde başlar:

```
import torch

x = torch.tensor([[1.0], [2.0], [3.0], [4.0]])
y = torch.tensor([[2.0], [4.0], [6.0], [8.0]])
```

Burada yaptığımız şey aslında oldukça basittir. Sisteme dört adet giriş ve bunlara karşılık gelen dört adet doğru çıktı veriyoruz. Bu noktada henüz öğrenme gerçekleşmemiştir. Ortada yalnızca veri vardır.

Makine öğrenmesinde veri, deneyimin yerine geçer. İnsan nasıl yaşadığı deneyimlerden öğreniyorsa, model de gördüğü örneklerden öğrenir.

Modeli Tanımlamak

Şimdi modele ihtiyacımız var.

Model, girdiyi alıp çıktı üreten matematiksel bir fonksiyondur. En basit haliyle doğrusal bir model kullanacağız:

$$y = wx + b$$

$$w$$

$$b$$

Burada:

- w ağırlığı temsil eder
- b bias değeridir
- Model bu iki değeri öğrenmeye çalışacaktır

Başlangıçta sistem doğru değerleri bilmez. W ve b rastgele başlar.

İşte öğrenmenin temel fikri burada ortaya çıkar:

Öğrenme, doğru ağırlıkları bulma sürecidir.

PyTorch'ta bu modeli şöyle tanımlarız:

model = torch.nn.Linear(1, 1)

Bu satır son derece küçüktür ama çok güçlüdür. Çünkü artık elimizde öğrenebilen bir sistem vardır.

Linear(1,1) ifadesi şunu söyler:

- Bir giriş al
- Bir çıktı üret
- Aradaki ilişkiyi öğren

Bu kadar.

Modern yapay zekâ sistemlerinin temeli aslında bu kadar basit bir fikre dayanır.

Model İlk Başta Hiçbir Şey Bilmez

Şimdi ilginç bir deney yapabiliriz. Model henüz eğitilmeden önce tahmin üretmesini isteyelim:

print(model(x))

Çıktılar büyük ihtimalle anlamsız olacaktır:

ensör([[0.53],
[1.12],
[1.70],
[2.29]])

Çünkü model henüz ilişkiyi öğrenmemiştir.

Bu çok önemli bir noktadır. İnsanlar çoğu zaman yapay zekânın “her şeyi bilen” bir sistem olduğunu düşünür. Oysa başlangıçta model tamamen bilgisizdir. Aslında model, rastgele tahmin yapan bir sistem olarak işe başlar. Öğrenme süreci tam da bu bilgisizlikten düzen üretme sürecidir.

Hata Kavramı

Peki model nasıl öğrenecek?

Cevap: hata sayesinde.

Modelin yaptığı tahmin ile gerçek değer arasındaki fark hesaplanır.

Örneğin:

- Gerçek değer: 8
- Tahmin: 5
- Hata: 3

Makine öğrenmesinin kalbinde bu fikir vardır:

Model yanlış yaptığını görür ve kendini düzeltir.

PyTorch’ta hata fonksiyonunu şöyle tanımlarız:

loss_function = torch.nn.MSELoss()

MSE burada “Mean Squared Error” anlamına gelir.

Yani:

- Hataları hesapla
- Karelerini al
- Ortalamasını bul

Neden kare alıyoruz?

Çünkü büyük hataları daha fazla cezalandırmak isteriz.

Örneğin:

- 1 birim hata küçük olabilir
- Ama 100 birim hata ciddi bir problemdir

Bu nedenle sistem büyük hatalara daha duyarlı hale getirilir.

Optimizasyon: Kendini Düzeltme Mekanizması

Şimdi modelin kendini düzeltmesi gerekiyor.

Burada optimizer devreye girer.

Optimizer'ın görevi şudur:

Hataları azaltacak şekilde ağırlıkları güncellemek.

PyTorch'ta çok yaygın kullanılan optimizerlardan biri SGD'dir:

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

Burada:

- SGD = Stochastic Gradient Descent
- lr = learning rate (öğrenme oranı)

Öğrenme oranı çok önemlidir.

Eğer çok büyük olursa model dengesiz davranır. Eğer çok küçük olursa öğrenme çok yavaş olur.

Bu durum biraz dağdan aşağı inmeye çalışan bir insan gibidir:

- Çok büyük adımlar → düşebilirsin
- Çok küçük adımlar → çok yavaş ilerlersin

Model de optimum noktayı bulmaya çalışır.

Öğrenme Döngüsü

Şimdi öğrenmenin gerçek mekanizmasına geldik.

Makine öğrenmesi aslında tekrar eden bir döngüdür:

1. Tahmin yap
2. Hatayı hesapla
3. Hatanın nedenini bul
4. Kendini düzelt
5. Tekrar dene

Kod şu şekildedir:

```
for epoch in range(1000):

    prediction = model(x)

    loss = loss_function(prediction, y)
```

```
optimizer.zero_grad()

loss.backward()

optimizer.step()
```

Bu birkaç satır aslında modern yapay zekânın temelidir.

Şimdi bunları tek tek inceleyelim.

Prediction: Modelin Tahmini

prediction = model(x)

Bu satırda model veriyi alır ve tahmin üretir.

Henüz doğru olmayabilir.

Ama önemli olan şey tahmin üretmesidir.

Çünkü öğrenme ancak tahmin varsa mümkündür.

Tahmin olmadan hata hesaplanamaz.

Loss: Hatanın Hesaplanması

loss = loss_function(prediction, y)

Burada modelin ne kadar yanlış yaptığı ölçülür.

Bu sayı öğrenmenin pusulasıdır.

Eğer hata büyükse model kötü performans gösteriyor demektir. Eğer hata küçülüyorsa model öğreniyor demektir. Makine öğrenmesinde ilerleme genellikle şu şekilde gözlenir:

- Başta büyük hata
- Zamanla azalan hata
- Sonunda dengelenen hata

Zero Grad: Geçmişi Temizlemek

optimizer.zero_grad()

PyTorch gradyanları biriktirdiği için her adımda bunları sıfırlamamız gerekir.

Sezgisel olarak bunu şöyle düşünebiliriz:

Model eski hataları temizleyip yeni hataya odaklanır.

Bu satır teknik görünse de öğrenmenin düzenli ilerlemesi için gereklidir.

Backward: Öğrenmenin Kalbi

Şimdi en önemli satıra geldik:

loss.backward()

Bu satır backpropagation işlemini başlatır.

Backpropagation'ın temel fikri şudur:

Hatanın kaynağını bul ve sistemi düzelt.

Model hangi ağırlığın ne kadar hataya neden olduğunu hesaplar.

Böylece sistem şunu öğrenir:

- Hangi parametre yanlış?
- Ne kadar değişmeli?
- Hangi yönde güncellenmeli?

Bu süreç calculus ve türev kavramlarına dayanır.

PyTorch'un en büyük avantajlarından biri de burada ortaya çıkar:

Türevleri otomatik hesaplar.

Bu mekanizmaya autograd denir.

Kullanıcı karmaşık matematik işlemleriyle uğraşmadan öğrenen sistemler kurabilir.

Step: Kendini Güncelleme

optimizer.step()

Bu satır ağırlıkları değiştirir.

Artık model önceki model değildir.

Çünkü deneyimden öğrenmiştir.

İşte öğrenmenin özü budur:

Sistem deneyim sonucunda değişir.

Eğer sistem değişmiyorsa öğrenme yoktur.

Öğrenmenin Gözlenmesi

Şimdi eğitim sırasında hatayı yazdırabiliriz:

```
for epoch in range(1000):
    prediction = model(x)
    loss = loss_function(prediction, y)
```

```
    optimizer.zero_grad()

    loss.backward()

    optimizer.step()

    if epoch % 100 == 0:
        print(epoch, loss.item())
```

Başlangıçta hata büyük olacaktır:

0 15.82
100 2.14
200 0.45
300 0.09

Zamanla hata küçülür.

Bu çok etkileyici bir andır.

Çünkü artık soyut bir fikir değil, gözle görülen bir öğrenme süreci vardır.

Model giderek doğru ilişkiyi keşfetmektedir.

Model Gerçekten Ne Öğrendi?

Eğitim bittikten sonra ağırlıkları inceleyebiliriz:

```
print(model.weight)
print(model.bias)
```

Muhtemelen şu değerlere yakın sonuçlar görürüz:

weight = 2.00
bias = 0.01

Bu olağanüstü önemlidir.

Çünkü model:

- “y = 2x” kuralını doğrudan verilmeden keşfetmiştir
- Yalnızca örneklerden öğrenmiştir
- Matematiksel ilişkiyi kendisi bulmuştur

Makine öğrenmesinin büyüsü burada ortaya çıkar.

Yeni Veri Üzerinde Test

Şimdi modele daha önce görmediği bir veri verelim:

test = torch.tensor([[10.0]])

print(model(test))

Muhtemelen şöyle bir sonuç alırız:

ensör([[20.01]])

Bu çok kritik bir noktadır.

Çünkü model ezber yapmamıştır.

Genelleme yapmıştır.

Makine öğrenmesinin gerçek amacı budur:

Görmediği durumlarda doğru davranabilmek.

Eğer sistem yalnızca eğitim verisini ezberliyorsa aslında öğrenmiş sayılmaz.

Gerçek öğrenme yeni durumlara uyum sağlayabilmektir.

İnsan Öğrenmesi ile Benzerlik

Bu süreci insan öğrenmesine benzetebiliriz.

Bir çocuk basketbol oynarken:

- İlk başta hata yapar
- Topu kaçırır
- Sonra kendini düzeltir
- Tekrar dener
- Zamanla daha iyi olur

Yapay sinir ağları da benzer biçimde çalışır.

Onlar da:

- Tahmin yapar
- Hata görür
- Kendini günceller
- Performansını geliştirir

Bu nedenle modern yapay zekâ sistemleri tamamen mekanik sistemler değildir.

Onlar deneyimle değişen davranış sistemleridir.

Bu Küçük Örnek Neden Çok Önemlidir?

Bu örnek son derece basit görünmektedir. Ancak modern yapay zekânın temel mantığını içerir.

Bugünkü:

- Büyük dil modelleri
- Görüntü tanıma sistemleri
- Otonom araçlar
- Robotik sistemler
- Ses tanıma uygulamaları

aynı temel prensibe dayanır.

Aradaki fark yalnızca ölçek farkıdır.

Burada:

- Bir ağırlık vardı
- Modern modellerde milyarlarca ağırlık vardır

Burada:

- Dört veri örneği vardı
- Büyük modellerde trilyonlarca örnek vardır

Ama temel döngü aynıdır:

1. Tahmin yap
2. Hata hesapla
3. Kendini düzelt
4. Tekrar et

Aslında modern yapay zekânın büyük kısmı bu döngünün devasa ölçekte uygulanmasından ibarettir.

PyTorch'un Gücü

PyTorch'un eğitim dünyasında bu kadar popüler olmasının nedeni yalnızca güçlü olması değildir.

Asıl nedeni şudur:

Öğrenmeyi görünür hale getirir.

Kullanıcı:

- Veriyi görebilir
- Modeli değiştirebilir
- Hatanın nasıl azaldığını izleyebilir
- Ağırlıkların nasıl güncellendiğini anlayabilir

Bu nedenle PyTorch yalnızca bir yazılım kütüphanesi değil; öğrenen sistemleri anlamayı kolaylaştıran bir düşünme aracıdır.

Ve belki de en önemli gerçek şudur:

Yapay zekâ sihir değildir.

O, verilerden örüntü öğrenen matematiksel sistemlerin ürünüdür.

Bu nedenle en basit PyTorch örneği bile, modern yapay zekânın mantığını anlamak için son derece güçlü bir başlangıçtır.

2.6. Hata, Kayıp ve İyileştirme: Bir Model Nasıl Öğrenir?

Öğrenen sistemlerin kalbine inmek istediğimizde karşımıza çıkan en temel üç kavram vardır: hata, kayıp ve iyileştirme. Bu üç kavram, bir modelin nasıl öğrendiğini açıklayan mekanizmanın iskeletini oluşturur. Aslında modern yapay zekâ sistemlerinin karmaşık görünen davranışlarının arkasında oldukça basit bir fikir vardır: sistem hata yapar, hatayı ölçer ve bu hatayı azaltmaya çalışır.

Bu bölümde öğrenme sürecini teknik detaylardan ziyade sezgisel bir düzeyde ele alacağız. Çünkü öğrenmenin özü, matematiksel formüllerden önce bir davranış döngüsüdür. Modelin neyi doğru ya da yanlış yaptığını anlaması ve bu bilgiyi kendini geliştirmek için kullanmasıdır.

Öğrenmenin Temel Sorusu

Bir modelin öğrenmesini anlamak için şu temel soruyu sormamız gerekir:

👉 Bir sistem nasıl daha iyi hale gelir?

Bu sorunun cevabı üç aşamalı bir süreç içerir:

- Bir tahmin yapılır
- Bu tahminin ne kadar yanlış olduğu ölçülür
- Model bu hatayı azaltacak şekilde değiştirilir

İşte bu üç adım, öğrenmenin temel döngüsünü oluşturur.

Hata Nedir?

Hata, bir modelin yaptığı tahmin ile gerçek değer arasındaki farktır. En basit haliyle şöyle ifade edilebilir:

👉 Hata = Gerçek değer – Tahmin edilen değer

Örneğin:

- Gerçek değer: 10
- Model tahmini: 7
- Hata: 3

Bu fark, modelin ne kadar yanlış yaptığını gösterir.

Hata, öğrenme sürecinin en kritik bileşenidir çünkü modelin neyi yanlış yaptığını anlamasını sağlar. Hata olmadan öğrenme mümkün değildir. Çünkü öğrenme, yanlışlardan türeyen bir süreçtir.

Hata Neden Önemlidir?

Bir sistemin gelişebilmesi için önce nerede başarısız olduğunu bilmesi gerekir. Hata bu bilgiyi sağlar.

Eğer bir model hiç hata yapmazsa:

- Öğrenemez
- Gelişemez
- Değişemez

Ancak gerçek sistemler her zaman hata yapar. Bu hata, sistemin kendini geliştirmesi için bir sinyaldir.

Bu nedenle hata, bir problem değil, bir geri bildirim mekanizmasıdır.

Kayıp Nedir?

Hata tek bir örnek için ölçülür. Ancak gerçek dünyada milyonlarca örnek vardır. Bu nedenle hataların genel bir ölçüsü gerekir. İşte bu noktada “kayıp” kavramı devreye girer.

👉 Kayıp = tüm hataların genel ölçüsü

Kayıp fonksiyonu, modelin ne kadar kötü performans gösterdiğini toplu olarak ifade eder.

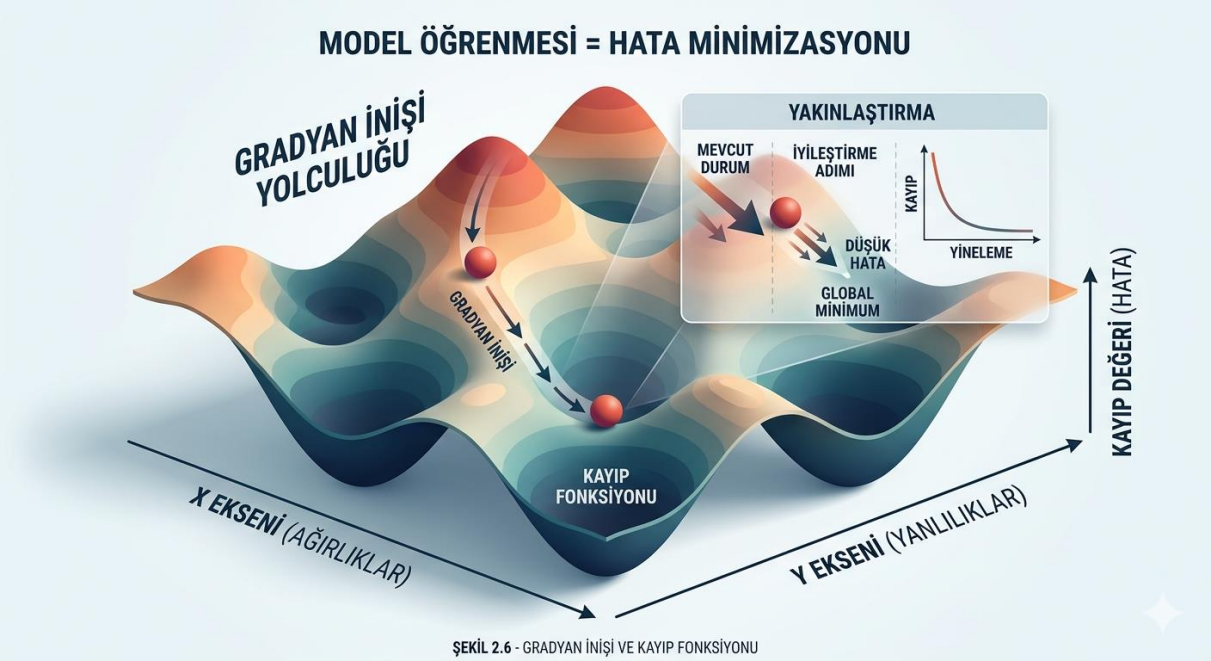
Örneğin:

- 100 tahmin yapıldı
- Her biri için hata hesaplandı
- Bu hatalar ortalandı

Ortaya çıkan değer “kayıp”tır.

Şekil 5 , yapay zekâ modellerinin nasıl öğrendiğini gösteren “gradyan inişi” sürecini anlatıyor. Yüzeydeki tepeler ve çukurlar, modelin yaptığı hata miktarını temsil eden kayıp fonksiyonudur. Amaç, yüksek hatalı bölgelerden aşağı doğru ilerleyerek en düşük hata noktasına, yani global minimuma ulaşmaktır. Model her adımda ağırlıklarını ve parametrelerini biraz değiştirerek daha doğru tahminler yapmayı öğrenir. Kısacası öğrenme süreci, hatayı sürekli azaltmaya çalışan matematiksel bir optimizasyon yolculuğudur.

Şekil 5. Model Öğrenmesi: Hata Minimizasyonu



Hata ve Kayıp Arasındaki Fark

Bu iki kavram sık sık karıştırılır. Ancak aralarında önemli bir fark vardır:

- Hata: tek bir örnek için ölçülür
- Kayıp: tüm veri seti için ölçülür

Bu farkı şu şekilde düşünebiliriz:

- Hata = tek bir sınav sorusu
- Kayıp = tüm sınavın sonucu

Modelin amacı tek bir soruyu değil, tüm sınavı iyi yapmaktır.

Neden Kayıp Fonksiyonu Kullanılır?

Çünkü modelin performansını ölçmek gerekir. Ölçüm olmadan iyileştirme mümkün değildir.

Kayıp fonksiyonu şunları sağlar:

- Modelin genel performansını ölçer

- Öğrenmenin yönünü belirler
- Optimizasyon sürecini yönlendirir

Kayıp fonksiyonu olmadan model, neyi geliştirmesi gerektiğini bilemez.

Öğrenme = Kayıp Azaltma Süreci

Makine öğrenmesinin en temel tanımı şudur:

👉 Öğrenme = kaybı minimize etme süreci

Bu süreçte model sürekli olarak şu soruyu sorar:

- "Hangi değişiklik hatamı azaltır?"

Bu sorunun cevabı modelin parametrelerini günceller.

İyileştirme Nedir?

İyileştirme, modelin hatalarını azaltacak şekilde kendini değiştirmesidir. Bu süreç genellikle "optimizasyon" olarak adlandırılır.

İyileştirme şu adımları içerir:

- Hatanın hesaplanması
- Hatanın yönünün belirlenmesi
- Model parametrelerinin güncellenmesi

Bu süreç tekrarlandıkça model daha iyi hale gelir.

Öğrenme Döngüsü

Bir modelin öğrenme süreci sürekli tekrar eden bir döngüdür:

1. Model tahmin yapar
2. Hata hesaplanır
3. Kayıp fonksiyonu değerlendirilir
4. Model güncellenir
5. Süreç tekrar eder

Bu döngü binlerce hatta milyonlarca kez tekrarlanabilir.

Gradient Descent: İyileştirmenin Mekanizması

Modelin nasıl iyileştirildiğini anlamak için "gradient descent" kavramına bakmak gerekir.

Bu yöntem şu prensibe dayanır:

👉 Hatanın en hızlı azaldığı yöne git

Model, hatayı azaltmak için parametrelerini küçük adımlarla değiştirir. Her adımda biraz daha iyi hale gelir.

Bu süreç şu şekilde düşünülebilir:

- Bir dağın tepesindesiniz
- Amaç en aşağıya inmek
- Her adımda eğimi takip ediyorsunuz

Bu metafor, öğrenmenin sezgisel karşılığıdır.

Öğrenme Neden Yavaş Bir Süreçtir?

Öğrenme ani bir süreç değildir. Çünkü:

- Veri çok büyüktür
- Hata yüzeyleri karmaşıktır
- En iyi çözüm hemen bulunamaz

Bu nedenle model küçük adımlarla ilerler.

Bu durum bir insanın öğrenmesine benzer:

- İlk denemede hata yapar
- Zamanla daha iyi olur
- Deneyim kazandıkça gelişir

Hata Olmadan Öğrenme Mümkün mü?

Hayır. Hata olmadan öğrenme mümkün değildir.

Çünkü:

👉 Hata = geri bildirim

Geri bildirim olmadan sistem kendini geliştiremez.

Bu nedenle modern yapay zekâ sistemleri bilinçli olarak hata yapar ve bu hatalardan öğrenir.

Kayıp Fonksiyonunun Rolü

Kayıp fonksiyonu, modelin “ne kadar kötü” olduğunu ölçer. Bu ölçüm olmadan iyileştirme mümkün değildir.

Farklı problemler için farklı kayıp fonksiyonları kullanılır:

- Regresyon problemleri
- Sınıflandırma problemleri
- Görüntü işleme problemleri

Her biri farklı hata ölçüm yöntemleri içerir.

Öğrenmenin Matematikten Bağımsız Sezgisi

Her ne kadar öğrenme matematiksel bir süreç olsa da, özü sezgiseldir.

Şöyle düşünebiliriz:

- Model bir şey dener
- Sonuç kötü çıkar
- Model bunu fark eder
- Bir sonraki denemede daha iyi yapar

Bu döngü, öğrenmenin temelidir.

İnsan Öğrenmesi ile Makine Öğrenmesi Arasındaki Benzerlik

İnsanlar da benzer şekilde öğrenir:

- Deneyim yaşar
- Hata yapar
- Düzeltir
- Gelişir

Makine öğrenmesi bu sürecin matematiksel modelidir.

İyileştirme Olmadan Öğrenme Olur mu?

Hayır. Çünkü öğrenme sadece hatayı görmek değildir. Aynı zamanda bu hatayı düzeltmektir.

İyileştirme olmadan sistem:

- Hatalarını bilir
- Ama değişmez

Bu nedenle öğrenme tamamlanmaz.

Sonuç: Öğrenmenin Üçlü Yapısı

Bu bölümde öğrenmenin üç temel bileşenini gördük:

- Hata: yanlışın ölçüsü

- Kayıp: toplam performans
- İyileştirme: değişim mekanizması

Bu üçü birlikte çalıştığında öğrenme gerçekleşir.

Sonuç olarak:

👉 Öğrenme = hata + geri bildirim + iyileştirme döngüsü

Bu döngü modern yapay zekânın temelidir.

Bu noktadan sonra artık sistemleri yalnızca programlayan değil, onların nasıl öğrendiğini anlayan bir bakış açısına sahip oluruz.

2.7. Ara Proje II: Görüntüden Anlam Çıkarmak (Basit Sınıflandırma)

Bu bölümde artık teorik çerçeveyi bir kenara bırakıp, öğrenen sistemlerin en temel ve en öğretici uygulamalarından birine geçiyoruz: görüntüden anlam çıkarmak. Bu problem, modern yapay zekânın en görünür alanlarından biridir çünkü bilgisayarların "görmesi" fikri hem sezgisel hem de çarpıcıdır. Ancak burada "görmek" aslında insan gözüyle görmek değildir; sayılarla temsil edilen bir veriyi anlamlı sınıflara ayırmaktır.

Bu ara projede amaç, karmaşık derin öğrenme modelleri kurmak değildir. Aksine, en basit haliyle bir sınıflandırma probleminin nasıl çalıştığını anlamaktır. Çünkü büyük modellerin temeli her zaman küçük ve anlaşılır örneklerdir.

Problemin Tanımı

Elimizde çok basit bir görev olduğunu düşünelim:

- Bir görüntü verisi var
- Bu görüntü ya "0" ya da "1" sınıfına ait
- Sistem bu görüntünün hangi sınıfa ait olduğunu tahmin edecek

Bu noktada problem aslında bir karar problemine dönüşür:

👉 "Bu görüntü hangi kategoriye ait?"

Örneğin:

- Siyah-beyaz bir görüntü
- Bir kısmı yoğun, bir kısmı boş
- Sistem bu görüntüyü analiz edip bir sınıfa atamak zorunda

Bu kadar basit bir yapı bile aslında gerçek dünyadaki birçok problemi temsil eder:

- Bir X-ray görüntüsünde hastalık tespiti

- Bir üretim hattında hatalı ürün tespiti
- Bir güvenlik kamerasında nesne tanıma

Görüntü Nedir? (Makine Perspektifi)

Bir bilgisayar için görüntü, insan gözüyle görülen bir “resim” değildir. Görüntü, sayılardan oluşan bir matristir.

Örneğin:

- 28x28 piksel bir görüntü düşünelim
- Her piksel 0 ile 255 arasında bir değere sahiptir

Bu durumda görüntü aslında şudur:

👉 28x28 boyutunda bir sayı matrisi

Bu matris düzleştirildiğinde bir vektör haline gelir. İşte modelin gördüğü şey budur.

Bu noktada önemli bir fark ortaya çıkar:

- İnsan: şekil görür
- Makine: sayı dizisi görür

Sınıflandırma Nedir?

Sınıflandırma, bir girdiyi belirli kategorilerden birine atama işlemidir.

Bu projede iki sınıf vardır:

- Sınıf 0
- Sınıf 1

Modelin görevi, verilen görüntünün hangi sınıfa ait olduğunu tahmin etmektir.

Matematiksel olarak:

👉 $x \rightarrow y$

- x: görüntü
- y: sınıf etiketi

Basit Model Mantığı

Bu projede karmaşık bir sinir ağı kullanmayacağız. Bunun yerine çok basit bir model düşünelim:

- Girdi: görüntü vektörü

- Çıktı: 0 veya 1

Modelin yaptığı şey şudur:

👉 Görüntüyü sayısal olarak analiz etmek ve bir skor üretmek

Bu skor daha sonra bir eşik değeri ile sınıfa dönüştürülür:

- skor > 0.5 → sınıf 1
- skor ≤ 0.5 → sınıf 0

Örnek: Nokta Yoğunluğu ile Sınıflandırma

Somut bir örnek düşünelim.

Elimizde iki tür görüntü olsun:

- Tür A: daha çok siyah piksel içeriyor
- Tür B: daha çok beyaz piksel içeriyor

Model şu basit kuralı öğrenebilir:

👉 Siyah piksel sayısı fazla ise sınıf 1

Bu durumda model aslında şu ilişkiyi öğrenmiş olur:

- yoğunluk → sınıf

Bu çok basit görünür, ama aslında öğrenmenin temelidir.

Veri Seti Nasıl Oluşur?

Bir sınıflandırma problemi için veri seti gerekir. Bu veri seti şunları içerir:

- Girdiler (görüntüler)
- Etiketler (doğru sınıflar)

Örneğin:

Görüntü	Etiket
img1	0
img2	1
img3	0

Model bu örnekleri kullanarak öğrenir.

Öğrenme Süreci

Modelin öğrenme süreci şu şekilde işler:

1. Bir görüntü alınır
2. Model tahmin yapar
3. Tahmin gerçek etiketle karşılaştırılır
4. Hata hesaplanır
5. Model güncellenir

Bu süreç tekrarlandıkça model daha iyi hale gelir.

Hata Örneği

Diyelim ki:

- Gerçek etiket: 1
- Model tahmini: 0

Bu durumda model yanlış yapmıştır.

Hata büyüktür ve model bu hatayı azaltmak zorundadır.

Kayıp Fonksiyonu

Tüm veri seti üzerinde modelin performansı “kayıp fonksiyonu” ile ölçülür.

Örneğin:

- 100 görüntü
- 30 yanlış tahmin
- Kayıp yüksek

Amaç:

👉 Bu kaybı azaltmak

Basit Bir Sezgisel Model

Bu projeyi daha sezgisel hale getirelim.

Model aslında şunu yapar:

- Görüntüdeki siyah piksel sayısını sayar
- Eğer fazla ise “1” der
- Az ise “0” der

Bu bile bir öğrenme biçimidir.

Gerçek Hayat Analojisi

Bu süreci bir insan örneğiyle düşünelim:

Bir çocuk meyveleri öğreniyor:

- Elma kırmızı → 1
- Muz sarı → 0

Başta hata yapar. Ama zamanla öğrenir.

Makine öğrenmesi de aynı süreci yaşar.

Modelin Öğrendiği Şey Nedir?

Model aslında görüntüleri "ezberlemez". Şunu öğrenir:

👉 Görüntü ile etiket arasındaki ilişki

Bu ilişki bir fonksiyondur:

x → y

Model bu fonksiyonu yaklaşık olarak öğrenir.

Neden Bu Kadar Önemli?

Bu basit proje, modern yapay zekânın temelini açıklar. Çünkü:

- Görüntü tanıma
- Nesne algılama
- Yüz tanıma
- Tıbbi görüntü analizi

hepsi aynı mantığa dayanır.

Vektör Perspektifi

Görüntü aslında bir vektördür:

- 28x28 = 784 boyut

Model bu 784 boyutlu vektörü alır ve bir karar üretir.

Bu şu anlama gelir:

👉 Görmek = vektör işlemek

Öğrenme Neden Çalışır?

Çünkü veri düzenlidir.

- Benzer görüntüler benzer vektörler oluşturur
- Bu benzerlik model tarafından öğrenilebilir

Bu nedenle model genelleme yapabilir.

Basit Ama Güçlü Fikir

Bu ara proje bize çok önemli bir gerçeği gösterir:

👉 Karmaşık görünen yapay zekâ sistemleri, aslında basit sınıflandırma problemlerinin birleşimidir

Sonuç

Bu projede şunu öğrendik:

- Görüntü = sayı matrisi
- Sınıflandırma = karar verme problemi
- Model = fonksiyon yaklaşımı
- Öğrenme = hata azaltma süreci

Bu basit örnek, yapay zekânın temel mantığını görünür hale getirir. Artık bir görüntüye baktığımızda sadece bir resim görmüyoruz. Aynı zamanda sayılardan oluşan bir yapı ve bu yapının öğrenilebilir bir model tarafından nasıl yorumlandığını da görebiliyoruz. Bu bakış açısı, kitabın ilerleyen bölümlerinde daha karmaşık sistemlere geçiş için temel zihinsel altyapıyı oluşturur.

BÖLÜM 3 — ROBOTİK SİSTEMLERİN TEMELLERİ

Bu bölümde artık öğrenen sistemlerin soyut yapısından çıkıp fiziksel dünyaya geçiyoruz. Robotik sistemler yalnızca algoritmalardan ibaret değildir; aynı zamanda gerçek dünyayla etkileşen, sensörler aracılığıyla veri toplayan ve aktüatörler aracılığıyla hareket üreten bütüncül yapılardır. Bu nedenle robotik, yazılım ile donanımın birbirinden ayrılmadığı, sürekli etkileşim halinde olduğu bir mühendislik alanı olarak düşünülmelidir.

Robotik sistemlerin temelinde üç ana bileşen bulunur: algılama, karar verme ve eylem üretme. Algılama aşamasında robot çevresinden veri toplar. Bu veriler kameralar, lidar sistemleri, sıcaklık sensörleri veya temas sensörleri gibi farklı fiziksel araçlar üzerinden elde edilir. Karar verme aşamasında bu veriler işlenir ve sistemin hangi davranışı sergilemesi gerektiği belirlenir. Bu aşamada yapay zekâ yöntemleri ve kontrol algoritmaları devreye girer. Son aşama olan eylem üretiminde ise robot motorlar, robot kolları veya tekerlek sistemleri aracılığıyla fiziksel dünyada hareket üretir.

Bu bölüm boyunca robotik sistemlerin nasıl yapılandığını, yazılım ile donanımın nasıl birlikte çalıştığını ve basit bir robotun nasıl davranış ürettiğini anlamaya odaklanacağız. Buradaki amaç robotu yalnızca hareket eden bir makine olarak görmek değil, çevresiyle sürekli etkileşim kuran bir sistem olarak kavramaktır.

3.1. Sensörler: Dünyayı Algılamak

Robotik sistemlerin en temel bileşenlerinden biri sensörlerdir. Çünkü bir robotun akıllı bir davranış sergileyebilmesi için önce çevresini algılaması gerekir. Algılama olmadan karar verme mümkün değildir ve karar verme olmadan da eylem üretilemez. Bu nedenle sensörler robotun dış dünyayla kurduğu ilk temas noktasıdır.

İnsanlar dünyayı görme, işitme, dokunma gibi duyularla algılar. Robotlar ise bu işlevi sensörler aracılığıyla gerçekleştirir. Ancak burada önemli bir fark vardır. İnsan algısı bütüncül ve sezgisel iken robot algısı sayısal ve parçalıdır. Bir robot için dünya sürekli akan bir deneyim değil, ölçülebilir veri parçalarından oluşan bir yapıdır.

Sensörler fiziksel dünyadaki büyüklükleri ölçülebilir elektriksel sinyallere dönüştüren cihazlardır. Sıcaklık, ışık, mesafe veya basınç gibi fiziksel büyüklükler sensörler aracılığıyla sayısal değerlere çevrilir. Örneğin bir sıcaklık sensörü ortamın ısısını ölçerek bunu bir sayı olarak ifade eder. Bir kamera ışığı dijital görüntüye çevirir. Bir ultrasonik sensör ise mesafeyi hesaplayarak robotun çevresi hakkında bilgi üretir. Bu dönüşüm sayesinde fiziksel dünya bilgisayarların işleyebileceği bir forma dönüşür.

> Sensörler robotlar için yalnızca veri toplayan araçlar değildir, aynı zamanda çevresel farkındalığın temelini oluştururlar. Bir robot sensörler olmadan nerede olduğunu bilemez, neyle karşılaştığını anlayamaz ve yaptığı hareketlerin sonucunu değerlendiremez. Bu nedenle sensörler robotik sistemlerde göz ve kulak işlevi görür.

Sensörlerin ürettiği veriler genellikle ham verilerdir. Bu veriler sayısal değerler, konum vektörleri veya görüntü matrisleri şeklinde olabilir. Ancak bu ham veriler tek başına anlamlı değildir. Örneğin bir sıcaklık sensörünün 35 derece ölçmesi tek başına bir karar üretmez. Ancak bu veri bir karar sistemiyle birleştiğinde fanın çalıştırılması gibi bir davranışa dönüşebilir. Bu nedenle sensör verisi her zaman bir işleme sürecinden geçmek zorundadır.

Gerçek dünyadan gelen veriler aynı zamanda gürültü içerir. Işık değişimleri, gölgeler, mekanik titreşimler veya çevresel faktörler ölçüm sonuçlarını etkileyebilir. Bu nedenle sensör verileri doğrudan kullanılmaz, filtreleme ve işleme süreçlerinden geçirilir. Bu süreç, fiziksel dünyanın belirsizliğini sayısal dünyada yönetilebilir hale getirir.

Sensörler ile başlayan bu süreç, aslında robotik sistemlerin temel döngüsünü oluşturur. Fiziksel dünya ölçülür, sayısal veriye dönüştürülür, bu veri işlenir ve anlamlı hale getirilerek karar üretimine aktarılır. Bu zincir olmadan robotik sistemlerin çalışması mümkün değildir.

3.2. Aktüatörler: Dünyaya Etki Etmek

Robotik sistemlerin ikinci temel bileşeni aktüatörlerdir. Eğer sensörler robotun dünyayı algılamasını sağlıyorsa, aktüatörler de bu dünyaya müdahale etmesini sağlar. Bir robotun gerçekten varlık gösterebilmesi yalnızca çevresini anlamasıyla değil, aynı zamanda çevresini değiştirebilmesiyle mümkündür.

> Aktüatörler kontrol sinyallerini fiziksel harekete dönüştüren mekanizmalardır. Bu hareket döndürme, itme, çekme veya yön değiştirme şeklinde olabilir. Bir elektrik motoru tekerlekleri döndürürken, bir servo motor robot kolunun hassas hareketlerini sağlar. Hidrolik sistemler ise yüksek kuvvet gerektiren endüstriyel uygulamalarda kullanılır.

Robotik sistemlerde sensörler ile aktüatörler arasında sürekli bir döngü vardır. Robot önce çevresini algılar, sonra karar verir ve ardından bu kararı fiziksel harekete dönüştürür. Bu süreç sürekli tekrar eder ve robotun çevreyle etkileşim içinde kalmasını sağlar.

Aktüatörler olmadan robotlar yalnızca veri toplayan pasif sistemler olurdu. Hiçbir hareket üretemez, dünyayı değiştiremez ve etkileşim kuramazlardı. Bu nedenle aktüatörler robotun bedeni olarak düşünülebilir.

Aktüatörler fiziksel sistemler oldukları için ideal davranmazlar. Sürtünme, gecikme, enerji tüketimi ve mekanik aşınma gibi gerçek dünya etkilerine sahiptirler. Bu nedenle robotik sistemlerde kontrol mekanizmaları yalnızca dijital değil, aynı zamanda fiziksel sınırlamaları da dikkate almak zorundadır.

Bir robot kolunun hareketi, bir motorun dönmesi veya bir aracın yön değiştirmesi her zaman bir kontrol sinyaline bağlıdır. Bu sinyal yapay zekâ veya kural tabanlı sistemler tarafından üretilir ve aktüatörler bu sinyali fiziksel harekete dönüştürür. Bu dönüşüm, robotik sistemlerin dijital kararlarını gerçek dünyada görünür hale getirir.

3.3. Geri Besleme: Kontrolün Kalbi

Robotik sistemlerin gerçekten çalışmasını sağlayan temel mekanizma geri beslemedir. Geri besleme, sistemin kendi çıktısını ölçerek yeniden girişe dahil etmesi sürecidir. Bu mekanizma olmadan robotlar açık çevrimli sistemler haline gelir ve çevreye uyum sağlayamaz.

> Geri besleme üç temel adım içerir. Sistem önce bir eylem üretir, ardından bu eylemin sonucunu ölçer ve son olarak bu ölçüme göre kendini düzeltir. Bu döngü sürekli tekrar eder ve sistemin kararlı bir şekilde çalışmasını sağlar.

Örneğin bir robot kol belirli bir konuma gitmek istediğinde önce hareket eder, sonra konumunu ölçer ve hedefle karşılaştırır. Eğer hata varsa bu hata düzeltilir. Aynı şekilde bir drone havada sabit kalmak istediğinde rüzgâr etkisiyle sapmalar oluşur ve geri besleme sistemi bu sapmaları sürekli düzeltir.

Geri besleme yalnızca kontrol sistemlerinde değil, öğrenme sistemlerinde de temel bir rol oynar. Makine öğrenmesi modelleri de tahmin yapar, hata hesaplar ve bu hatayı minimize edecek şekilde kendini günceller. Bu nedenle öğrenme süreci aslında bir geri besleme döngüsüdür.

Geri besleme olmadan sistemler sabit kalır ve değişen çevre koşullarına uyum sağlayamaz. Bu nedenle modern robotik sistemler yalnızca programlanmış davranışlar değil, sürekli kendini düzelten dinamik yapılar olarak tasarlanır.

Sonuç

Robotik sistemler sensörler, aktüatörler ve geri besleme mekanizmalarının birleşiminden oluşur. Sensörler dünyayı algılar, aktüatörler dünyaya etki eder ve geri besleme bu iki süreci sürekli dengede tutar. Bu yapı olmadan robotik sistemlerin ne öğrenmesi ne de davranış üretmesi mümkündür.

Bu bölümün temel mesajı şudur: robotik sistemler yalnızca makineler değildir, aynı zamanda sürekli algılayan, karar veren ve hareket eden dinamik yapılardır. Bu yapıların merkezinde ise fiziksel dünya ile sayısal dünya arasında kurulan sürekli etkileşim yer alır.

3.4. Geri Besleme Nedir?

Geri besleme, bir sistemin kendi çıktısını ölçerek bu çıktıyı tekrar girişe dahil etmesi sürecidir. Daha basit bir ifadeyle sistem yaptığı şeyi görür, değerlendirir ve kendini düzeltir. Bu mekanizma, robotik ve kontrol sistemlerinin temel çalışma prensibini oluşturur çünkü bir sistemin yalnızca eylem üretmesi yeterli değildir, aynı zamanda bu eylemin sonucunu da anlaması gerekir.

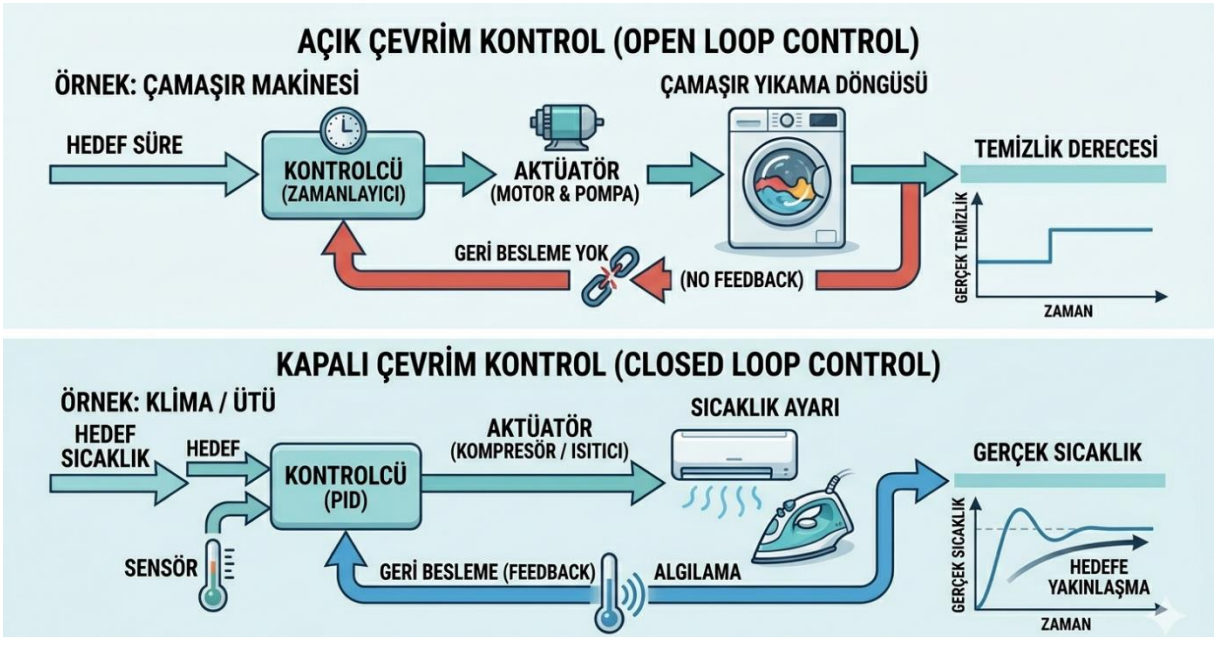
Geri besleme süreci üç temel bileşen üzerinden çalışır. İlk olarak sistem bir çıktı üretir. Bu çıktı bir hareket, bir sinyal ya da fiziksel bir değişim olabilir. İkinci aşamada bu çıktı ölçülür ve sistemin gerçek durumu belirlenir. Üçüncü aşamada ise ölçülen değer ile hedeflenen değer arasındaki fark hesaplanır ve sistem bu farka göre kendini düzeltir. Bu üçlü yapı sürekli olarak tekrar eder ve sistemin davranışı zaman içinde daha kararlı hale gelir.

Örneğin bir robot kol bir nesneyi tutmaya çalışıyorsa önce hareket eder, ardından sensörler nesnenin konumunu ölçer ve son olarak hata varsa bu hata düzeltilir. Bu süreç tek seferlik değildir, milisaniyeler düzeyinde sürekli tekrar eden bir döngüdür. Bu tekrar, robotun çevresiyle uyum içinde çalışmasını sağlar.

3.5. Açık Çevrim ve Kapalı Çevrim

Geri beslemeyi anlamanın en iyi yolu açık çevrim ve kapalı çevrim sistemleri karşılaştırmaktır. Açık çevrim sistemlerde sistem yalnızca çıktı üretir, ancak bu çıktının sonucunu ölçmez ve düzeltme yapmaz. Bu nedenle sistem basittir ama dış koşullara karşı oldukça kırılgandır.

Şekil 6. Kontrol Tipleri



Şekil 6 bize, açık çevrim ve kapalı çevrim kontrol sistemleri arasındaki farkı gösteriyor. Açık çevrim kontrolde sistem yalnızca verilen komuta göre çalışır ve sonucu ölçmez; örneğin bir çamaşır makinesi belirlenen süre boyunca çalışır ama çamaşırın gerçekten ne kadar temiz olduğunu kontrol etmez. Kapalı çevrim kontrolde ise sensörler sürekli ölçüm yapar ve sistem geri besleme sayesinde kendini düzeltir. Klima veya ütü örneğinde olduğu gibi, gerçek sıcaklık sürekli ölçülür ve hedef değere yaklaşacak şekilde ayarlamalar yapılır. Modern robotik sistemlerin çoğu, çevreye uyum sağlayabilmek için kapalı çevrim mantığıyla çalışır.

Örneğin yalnızca süreye göre çalışan bir çamaşır makinesi açık çevrim sistemine örnektir. Aynı şekilde sabit hızda çalışan bir motor da çevresel değişimlerden etkilenmeden çalışır. Bu sistemler tasarım açısından kolaydır ancak hata durumunda kendini düzeltemez.

Kapalı çevrim sistemlerde ise durum farklıdır. Bu sistemler çıktıyı ölçer, hedefle karşılaştırır ve ortaya çıkan hatayı düzeltir. Bu yapı sayesinde sistem çevresel değişimlere uyum sağlayabilir. Termostatlı ısıtıcılar, otonom araçlar ve robot kollar kapalı çevrim sistemlere örnektir.

Modern robotik sistemlerin büyük çoğunluğu kapalı çevrim mantığıyla çalışır çünkü gerçek dünya sürekli değişen ve belirsizlik içeren bir yapıya sahiptir.

3.6. Geri Besleme Neden “Kalp”tir?

Geri besleme robotik sistemlerin kalbidir çünkü sistemin canlı ve uyarlanabilir olmasını sağlar. Geri besleme olmadan sistem yalnızca sabit kurallara göre çalışan bir makineye dönüşür.

Geri besleme üç temel özellik kazandırır. Birincisi stabilitedir, yani sistemin dengede kalmasını sağlar. İkincisi doğruluktur, yani sistemin hedefe daha yakın sonuçlar üretmesini sağlar. Üçüncüsü ise uyarlanabilirliktir, yani sistem değişen çevre koşullarına uyum sağlar.

Geri besleme olmayan bir sistemde hatalar zamanla birikir ve sistem giderek hedefinden uzaklaşır. Ancak geri besleme ile sistem sürekli kendini düzeltir ve bu sapmalar kontrol altında tutulur. Bu nedenle geri besleme, robotik sistemlerde yalnızca bir kontrol mekanizması değil, aynı zamanda sistemin varlığını sürdüren temel yapı olarak görülür.

Örnek 1: Termostat Sistemi

Termostat sistemi geri beslemenin en temel örneklerinden biridir. Sistem belirli bir hedef sıcaklığa ulaşmayı amaçlar ve bu hedefi sürekli olarak ölçülen değerle karşılaştırır.

Örneğin hedef sıcaklık 22 derece olarak belirlenmişse sistem ortam sıcaklığını sürekli ölçer. Eğer sıcaklık 20 dereceye düşerse ısıtıcı devreye girer, eğer 24 dereceye çıkarsa ısıtıcı kapanır. Bu süreç sürekli tekrar eder ve ortam sıcaklığı hedef değer etrafında dengelenir.

Bu yapı oldukça basittir ancak geri beslemenin temel mantığını çok net bir şekilde gösterir. Ölçüm, karşılaştırma ve düzeltme döngüsü burada açık şekilde görülür.

Örnek 2: Drone Stabilizasyonu

Bir drone havada sabit kalmak istediğinde sürekli olarak çevresel etkilere maruz kalır. Rüzgâr, hava akımları ve titreşimler drone'un dengesini bozar. Bu noktada geri besleme sistemi devreye girer.

Drone içindeki gyro sensörleri eğim değişimlerini sürekli ölçer. Sistem bu ölçümleri referans değerlerle karşılaştırır ve sapma varsa motor hızlarını değiştirerek dengeyi sağlar. Bu süreç saniyenin çok küçük dilimlerinde sürekli tekrar eder.

Geri besleme olmasaydı drone sabit kalamaz, sürekli savrularak kontrolünü kaybederdi. Bu nedenle stabil uçuş tamamen geri besleme döngüsüne bağlıdır.

Örnek 3: Robot Kol

Bir robot kol bir nesneyi kavramaya çalıştığında çok hassas bir kontrol süreci gerekir. Çünkü küçük bir hata bile nesnenin düşmesine neden olabilir.

Bu süreçte kamera veya sensörler nesnenin konumunu belirler, robot kol kendi pozisyonunu ölçer ve iki değer arasındaki fark hesaplanır. Eğer hata varsa motorlar çok küçük düzeltmeler yapar. Bu düzeltmeler sürekli olarak tekrar eder.

Bu sayede robot kol nesneyi hassas bir şekilde kavrayabilir. Bu hassasiyetin temelinde yine geri besleme döngüsü vardır.

Geri Besleme Döngüsü

Geri besleme sürekli tekrar eden bir döngü şeklinde çalışır. Sistem önce bir eylem üretir, ardından çevrede oluşan değişimi ölçer, bu ölçümden hata hesaplar ve son olarak sistemi bu hatayı azaltacak şekilde günceller.

Bu döngü kesintisiz şekilde devam eder. Robotik sistemlerin dinamik ve çevreye uyumlu olmasını sağlayan temel yapı budur.

Hata Kavramı

Geri besleme sisteminin merkezinde hata kavramı bulunur. Hata, istenen durum ile gerçek durum arasındaki farktır. Bu fark ne kadar büyükse sistemin düzeltme ihtiyacı o kadar fazladır.

Örneğin hedef değer 10 iken sistem 7 ölçüyorsa hata 3'tür. Sistem bu hatayı azaltmak için kendini günceller. Bu süreç tekrar ettikçe hata küçülür ve sistem hedefe yaklaşır.

Negatif Geri Besleme

Negatif geri besleme sistemin dengeye gelmesini sağlar. Burada amaç hatayı azaltmaktır. Sıcaklık düştüğünde ısıtıcı açılır, sıcaklık yükseldiğinde kapanır. Bu yapı sistemin belirli bir aralıkta stabil kalmasını sağlar.

Pozitif Geri Besleme

Pozitif geri besleme ise hatayı büyüten bir yapıdır. Örneğin mikrofon ve hoparlör birbirine çok yakın olduğunda ses sürekli büyür ve kontrolsüz bir döngü oluşur. Bu tür sistemler genellikle kararsızdır.

Robotikte Geri Besleme Olmazsa Ne Olur?

Geri besleme olmayan robotlar çevresel değişimlere uyum sağlayamaz. Hatalar birikir ve sistem hedefinden uzaklaşır. Bu durumda robot sabit ama kırılgan bir davranış sergiler.

Geri Besleme ve Öğrenme

Geri besleme yalnızca kontrol mekanizması değildir, aynı zamanda öğrenmenin temelidir. Makine öğrenmesi modelleri de tahmin yapar, hata hesaplar ve bu hatayı azaltacak şekilde kendini günceller. Bu nedenle öğrenme süreci aslında sürekli bir geri besleme döngüsüdür.

Sonuç

Geri besleme robotik sistemlerin kalbidir çünkü algı, karar ve eylem arasındaki sürekli düzeltme mekanizmasını oluşturur. Sensörler veri üretir, aktüatörler hareket eder, geri besleme ise bu iki süreci anlamlı bir döngüye bağlar. Bu yapı olmadan robotik sistemlerin ne öğrenmesi ne de uyum sağlaması mümkündür.

Geri beslemeyi anlamak, bir sistemin neden akıllı davrandığını anlamaktır.

3.7. Basit Kontrol Sistemi ve PID Mantığı

Kontrol sistemlerini anlamak için önce temel problemi tanımlayalım: Bir sistemin istenen bir değeri takip etmesi nasıl sağlanır? Robotikte ve mühendislikte bu problem son derece yaygındır. Örneğin:

- Bir robot kolun belirli bir konuma gitmesi
- Bir drone'un havada dengede kalması
- Bir aracın hızını 60 km/s'de sabit tutması
- Bir odanın sıcaklığının belirli bir seviyede tutulması

Bu tür problemlerde sistemin ulaşmak istediği hedef değere "setpoint" denir. Sistem sürekli olarak mevcut durumu ölçer ve bunu hedefle karşılaştırır. Aradaki fark ise "hata" olarak adlandırılır.

Örneğin bir aracın hedef hızı 60 km/s olsun. Eğer araç 50 km/s hızla gidiyorsa hata:

$$Hata = Hedef - Ger\overset{\backslash c}{c}ek\ Değer = 60 - 50 = 10$$

olacaktır.

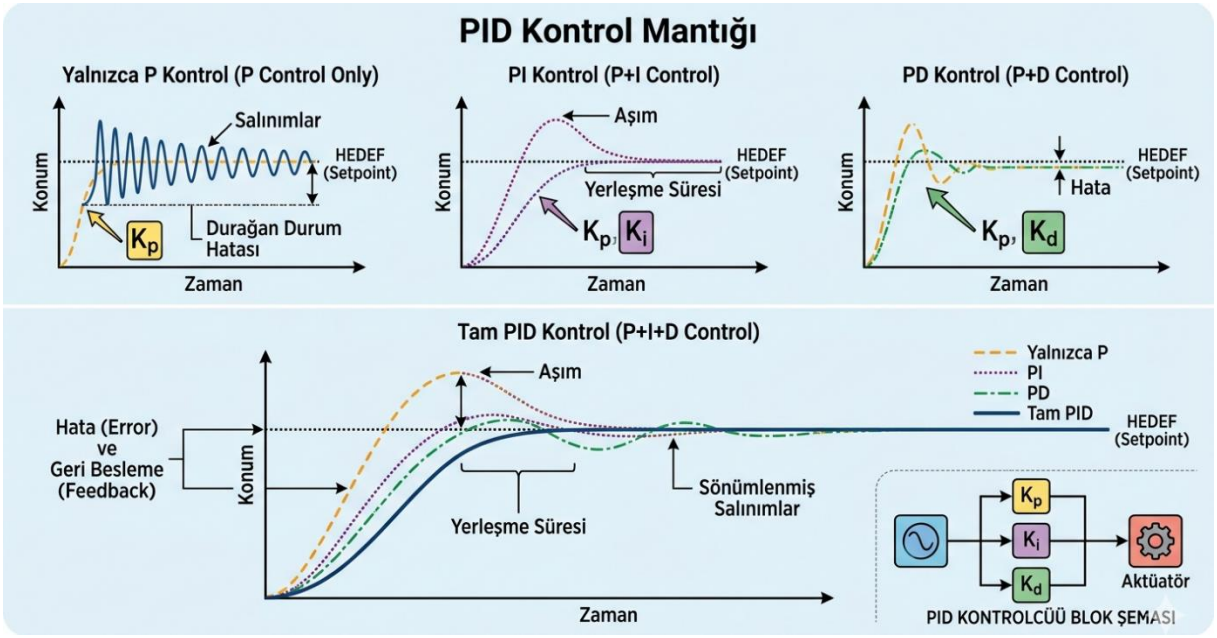
Kontrol sisteminin görevi bu hatayı mümkün olduğunca küçültmektir. İşte PID kontrolü tam olarak bu amaçla geliştirilmiş en önemli yöntemlerden biridir.

PID şu üç bileşenden oluşur:

- P → Proportional (Oransal)
- I → Integral (İntegral)
- D → Derivative (Türevsel)

Bu üç mekanizma birlikte çalışarak sistemin dengeli ve kararlı davranmasını sağlar. Şekil 7 bir sistemin hedeflenen değere (Setpoint) ulaştırılması sürecinde kullanılan **PID kontrol mantığının** farklı varyasyonlarını ve bunların sistem üzerindeki etkilerini karşılaştırmalı olarak özetlemektedir.

Şekil 7. PID Kontrol Mnatığı



P: Oransal Kontrol

Oransal kontrol en temel fikirdir. Mantığı şudur:

Hata ne kadar büyükse, tepki de o kadar büyük olsun.

Eğer araç hedef hızdan çok uzaksa sistem motora daha fazla güç verir. Hata küçüldükçe tepki de küçülür.

Bu yaklaşım hızlı tepki sağlar ancak tek başına yeterli değildir. Çünkü sistem bazen hedefe tam ulaşamaz ve küçük bir hata sürekli kalabilir.

I: İntegral Kontrol

İntegral kısmı geçmişte biriken hataları dikkate alır.

Örneğin araç sürekli olarak hedef hızın biraz altında kalıyorsa, integral kontrol bu küçük hataları zaman içinde toplar ve sistemi hedefe doğru iter.

Bu nedenle integral bileşeni şu problemi çözer:

Sürekli kalan küçük hataları yok etmek.

Ancak integral çok güçlü olursa sistem aşırı tepki verebilir ve salınımlar oluşabilir.

D: Türevsel Kontrol

Türevsel kontrol ise hatanın değişim hızına bakar.

Sistem hedefe çok hızlı yaklaşıyorsa, derivative bileşeni "fren etkisi" oluşturur. Böylece sistemin aşırı ileri gitmesi engellenir.

Bu mekanizma özellikle drone, robot kol ve otonom araçlarda çok önemlidir. Çünkü fiziksel sistemler yalnızca hedefe ulaşmak değil, aynı zamanda kararlı davranmak zorundadır.

PID mantığını sezgisel olarak şöyle düşünebiliriz:

- P → Şu anki hataya tepki verir
- I → Geçmiş hataları düzeltir
- D → Gelecekteki taşmaları önlemeye çalışır

Bu nedenle PID kontrolü aslında oldukça akıllı bir denge mekanizmasıdır.

Modern robotikte PID hâlâ çok yaygın kullanılmaktadır. Çünkü birçok fiziksel sistemde hızlı, anlaşılır ve güvenilir kontrol sağlar. Bir robotun dengede yürümesi, motorların hassas hareket etmesi ya da bir drone'un rüzgâr altında sabit kalması çoğu zaman PID benzeri kontrol mantıklarına dayanır.

Bugünkü gelişmiş yapay zekâ sistemlerinde bile klasik kontrol teorisi önemini korumaktadır. Çünkü öğrenen sistemler ne kadar güçlü olursa olsun, fiziksel dünyada hareket eden robotların hâlâ denge, hız, konum ve kararlılık problemlerini çözmesi gerekir. PID, bu problemlerin çözümünde mühendisliğin en temel araçlarından biridir.

PID'nin Sezgisel Anlamı

PID kontrolü şu üç soruya cevap verir:

- Şu anda ne kadar yanlışım? (P)
- Geçmişte ne kadar yanlış yaptım? (I)
- Yanlışım nasıl değişiyor? (D)

Bu üç bilgi birleştiğinde sistem çok daha dengeli hale gelir.

Örnek 1: Oda Sıcaklığı Kontrolü

Bir ısıtma sistemi düşünelim:

- Hedef: 22°C
- Gerçek: 18°C

P bileşeni:
Sadece 4°C farkı görür ve ısıtıcıyı açar.

I bileşeni:
Eğer sistem uzun süre düşük kalıyorsa daha fazla ısı verir.

D bileşeni:
Sıcaklık hızlı yükseliyorsa sistemi yavaşlatır.

Sonuç:

👉 Stabil sıcaklık kontrolü sağlanır.

Örnek 2: Drone Dengeleme

Bir drone havada sabit kalmak ister.

- Rüzgâr drone'u iter
- Drone eğilir
- Sensör bunu algılar

P: anlık düzeltme yapar
I: sürekli kaymayı düzeltir
D: ani hareketleri yumuşatır

Bu üçü birlikte drone'un stabil kalmasını sağlar.

Örnek 3: Robot Kol Pozisyonu

Bir robot kol hedef pozisyona gitmek istiyor.

- İlk başta büyük hata vardır
- P hızlı hareket eder
- I küçük sapmaları düzeltir
- D aşırı hareketi engeller

Sonuç: düzgün ve hassas hareket

PID Olmadan Sistem Ne Olur?

Eğer PID kullanılmazsa:

- Sistem aşırı salınım yapar
- Hedefi aşar
- Kararsız hale gelir

Örneğin:

- Termostat sürekli aç-kapa yapar
- Drone sürekli sallanır
- Robot kol hedefi kaçırır

PID = İnsan Davranışı Analojisi

PID kontrolünü insan davranışına benzetebiliriz:

- P → refleks
- I → deneyim
- D → öngörü

Örneğin yürürken:

- Ayağın takılır → hızlı tepki (P)
- Sürekli aynı yerde takılıyorsan dikkat edersin (I)
- Denge kaybını hissedip önceden düzeltirsin (D)

PID'nin Gücü Nedir?

PID'nin gücü şuradan gelir:

👉 Basit ama etkili olması

Karmaşık yapay zekâ modelleri olmadan bile birçok sistem PID ile kontrol edilebilir.

Bu nedenle:

- Endüstride
- Robotikte
- Havacılıkta

çok yaygın kullanılır.

PID ve Öğrenme İlişkisi

PID aslında bir öğrenme sistemi değildir ama öğrenmeye çok benzer.

Çünkü:

- Hata ölçer
- Tepki verir
- Sistemi düzeltir

Bu yapı, öğrenmenin temel döngüsüne çok benzer.

PID'nin Sınırlamaları

PID güçlüdür ama sınırsız değildir:

- Karmaşık sistemlerde zorlanır
- Çok değişkenli problemlerde yetersiz kalabilir
- Dinamik ortamlarda ayar gerektirir

Bu nedenle modern sistemlerde PID, yapay zekâ ile birlikte kullanılır.

PID Ayarlama Problemi

PID'nin en zor kısmı parametre ayarlamadır:

- P çok yüksek → sistem salınır
- I çok yüksek → sistem yavaşlar
- D çok yüksek → sistem aşırı hassas olur

Bu nedenle doğru dengeyi bulmak kritiktir.

Modern Sistemlerde PID

Günümüzde PID tek başına değil, öğrenen sistemlerle birlikte kullanılır:

- Derin öğrenme + PID
- Reinforcement learning + PID
- Adaptif kontrol + PID

Bu kombinasyonlar daha güçlü sistemler üretir.

Sezgisel Özet

PID'yi en basit şekilde şöyle düşünebiliriz:

- P → "şu an ne yanlış?"
- I → "hep yanlış mı yapıyorum?"
- D → "yanlışım büyüyor mu?"

Bu üç soru birlikte kontrol sağlar.

Sonuç

Basit kontrol sistemleri, robotik dünyasında çok derin bir rol oynar. PID kontrolü, yalnızca bir mühendislik aracı değil, aynı zamanda sistemlerin nasıl dengede kaldığını açıklayan bir düşünme modelidir.

Bu bölümün temel mesajı şudur:

👉 Kontrol = hata yönetimi
👉 PID = hatayı anlama, biriktirme ve öngörme sistemi
👉 Robotik sistem = sürekli denge arayan bir yapı

PID'yi anlamak, bir robotun neden stabil çalıştığını anlamaktır.

3.8. Fiziksel ve Sayısal Dünya Arasında

Modern robotik ve yapay zekâ sistemlerini anlamanın en derin katmanlarından biri, fiziksel dünya ile sayısal dünya arasındaki ilişkiyi kavramaktır. Çünkü robotlar aslında bu iki dünya arasında sürekli gidip gelen varlıklardır. Bir yanda sürekli değişen, gürültülü, kesintisiz ve kaotik fiziksel gerçeklik vardır; diğer yanda ise bu gerçekliğin sayılarla temsil edildiği, kesikli, hesaplanabilir ve soyut bir dijital dünya bulunur.

Bu iki dünya arasındaki dönüşüm, robotik sistemlerin varlık sebebidir. Eğer fiziksel dünya sayısal dünyaya çevrilemezse sensörler işe yaramaz; eğer sayısal dünya tekrar fiziksel dünyaya dönüştürülemezse aktüatörler anlamsız kalır. Bu nedenle robotik sistemler, iki yönlü bir çeviri makinesi gibi çalışır: dünyayı sayıya çevirir, sayıyı dünyaya geri döndürür.

Fiziksel Dünya Nedir?

Fiziksel dünya, insanın ve robotun içinde yaşadığı gerçekliktir. Bu dünya:

- Sürekli değişir
- Kesintisizdir
- Gürültülüdür
- Ölçülemez gibi görünen belirsizlikler içerir

Örneğin:

- Bir odadaki sıcaklık sürekli değişir
- Işık miktarı saniyeden saniyeye farklıdır
- Nesneler hareket eder
- Ses dalgaları yayılır

Bu dünya matematiksel olarak doğrudan işlenemez. Çünkü fiziksel gerçeklik "sonsuz çözünürlüklüdür".

Sayısal Dünya Nedir?

Sayısal dünya ise bilgisayarların anladığı dünyadır. Bu dünya:

- Sayılardan oluşur
- Kesiklidir
- Ölçülebilir
- İşlenebilir

Örneğin:

- 25°C sıcaklık
- 1280x720 piksel görüntü
- 0.75 olasılık değeri
- (x, y, z) konum vektörü

Sayısal dünya, fiziksel dünyanın bir temsilidir.

Temel Problem: Çeviri Problemi

Robotik sistemlerin en temel problemi şudur:

👉 Fiziksel dünya nasıl sayısal dünyaya çevrilir ve tekrar fiziksel dünyaya geri döndürülür?

Bu problem iki yönlüdür:

1. Algılama (Physical → Digital)
2. Eylem (Digital → Physical)

1. Fizikselden Sayısala: Algılama

Bu dönüşüm sensörler aracılığıyla gerçekleşir.

Örneğin:

- Kamera → görüntüyü sayıya çevirir
- Mikrofon → sesi örnekler
- Lidar → mesafeyi ölçer
- Termometre → sıcaklığı sayıya çevirir

Bu süreçte fiziksel gerçeklik “örneklenir”.

Örnek: Kamera Görüntüsü

Bir kamera aslında şunu yapar:

- Işığı alır
- Piksel matrisine dönüştürür
- Sayısal veri üretir

Örneğin:

- 1920x1080 görüntü
- Her piksel 0–255 arası değer

Bu durumda fiziksel sahne artık bir matris olur.

Örnek: Ses Dönüşümü

Bir mikrofon:

- Ses dalgalarını alır
- Örnekleme yapar
- Dijital sinyale çevirir

Bu süreçte sürekli bir dalga, sayısal bir diziye dönüşür.

2. Sayısaldan Fiziksele: Eylem

Bu dönüşüm aktüatörler aracılığıyla gerçekleşir.

Örneğin:

- Motor → elektrik sinyalini harekete çevirir
- Servo → açısal hareket üretir
- Hidrolik sistem → kuvvet üretir

Burada dijital kararlar fiziksel dünyada karşılık bulur.

Örnek: Robot Kol

Bir robot kol şu süreci takip eder:

1. Sayısal sistem “90 derece dön” kararı verir
2. Bu karar motora iletilir
3. Motor fiziksel hareket üretir
4. Kol gerçek dünyada döner

Bu süreç sayısal kararın fiziksel etkiye dönüşmesidir.

Temel Gerilim: Süreklilik vs Kesiklilik

Fiziksel ve sayısal dünya arasındaki en temel farklardan biri sürekliliktir.

- Fiziksel dünya: sürekli
- Sayısal dünya: kesikli

Örneğin:

- Gerçek sıcaklık 23.4567°C olabilir
- Bilgisayar bunu 23°C olarak temsil eder

Bu fark “yaklaşım hatası” yaratır.

Örnek: Ölçüm Hatası

Bir sensör şunu ölçer:

- Gerçek mesafe: 10.237 cm
- Ölçülen: 10.2 cm

Bu küçük fark bile robotik sistemlerde önemlidir.

Gürültü Problemi

Fiziksel dünya her zaman gürültülüdür.

Gürültü nedir?

👉 Ölçümdeki rastgele sapmalar

Örneğin:

- Kamera ışık değişimlerinden etkilenir
- Sensörler titreşimden etkilenir
- Ortam koşulları ölçümü bozar

Sayısal dünya ise temizdir. Bu nedenle filtreleme gerekir.

Filtreleme: Sayısal Dünyanın Temizlenmesi

Robotik sistemlerde sensör verisi doğrudan kullanılmaz.

Bunun yerine:

- Ortalama alma
- Kalman filtreleri
- Gürültü azaltma teknikleri

kullanılır.

Amaç:

👉 Fiziksel dünyanın gürültüsünü azaltmak

Dijital Temsil Nedir?

Dijital temsil, fiziksel dünyanın sayılarla ifade edilmesidir.

Örneğin:

- Bir nesnenin konumu → (x, y, z)
- Bir görüntü → matris
- Bir hız → skaler değer

Bu temsil olmadan yapay zekâ çalışamaz.

Örnek: Otonom Araç

Bir otonom araçta süreç şöyledir:

1. Kamera çevreyi algılar
2. Radar mesafe ölçer
3. Lidar derinlik üretir
4. Sistem karar verir
5. Direksiyon hareket eder

Burada fiziksel dünya sürekli sayısal modele çevrilir.

Sayısal Dünya = Model

Yapay zekâ sistemlerinde sayısal dünya aslında bir modeldir.

Bu model:

- Gerçekliği temsil eder
- Tahmin üretir
- Karar verir

Ancak hiçbir model gerçek dünyanın kendisi değildir.

Model Hatası

Model ile gerçek dünya arasında her zaman fark vardır.

Bu fark:

👉 Model hatası

Örneğin:

- Model nesneyi 1 metre uzaklıkta sanır
- Gerçekte 0.9 metredir

Bu fark kritik olabilir.

Geri Besleme ile Denge

Bu fark geri besleme ile düzeltilir.

Sistem:

- Ölçer
- Karşılaştırır
- Düzeltir

Bu döngü sayesinde model gerçek dünyaya yaklaşır.

Fiziksel Dünya Sürekli Değişir

En önemli problem şudur:

👉 Fiziksel dünya sabit değildir

Örneğin:

- Işık değişir
- Nesneler hareket eder
- Ortam değişir

Bu nedenle sayısal model sürekli güncellenmelidir.

Adaptasyon Gerekliliği

Robotik sistemler adaptif olmalıdır.

Yani:

- Yeni durumlara uyum sağlar
- Modelini günceller
- Öğrenir

Bu adaptasyon, fiziksel ve sayısal dünya arasındaki köprüdür.

Sezgisel Analiz

Bu ilişkiyi şöyle düşünebiliriz:

- Fiziksel dünya = gerçeklik
- Sayısal dünya = harita

Ama harita asla gerçeklik değildir. Sadece bir temsilidir.

Harita–Arazi Problemi

Bu klasik bir problemdir:

👉 Harita ≠ Arazi

Robotlar haritayı kullanarak araziyi anlamaya çalışır.

Öğrenme Neden Gerekli?

Çünkü:

- Fiziksel dünya değişir
- Sensörler hatalıdır
- Modeller eksiktir

Bu nedenle sistemler öğrenmek zorundadır.

Sonuç

Fiziksel ve sayısal dünya arasındaki ilişki, robotik sistemlerin temelidir. Bu iki dünya arasında sürekli bir çeviri süreci vardır ve bu süreç olmadan hiçbir akıllı sistem çalışamaz.

Bu bölümün temel mesajı şudur:

👉 Fiziksel dünya = gerçeklik
👉 Sayısal dünya = temsil
👉 Robotik sistem = bu iki dünya arasındaki sürekli çeviri makinesi

Bu bakış açısı, robotları yalnızca makineler olarak değil, iki farklı gerçeklik arasında çalışan sistemler olarak anlamamızı sağlar.

3.9. Ara Proje III: Çizgi İzleyen ve Engelden Kaçan Sistem

Bu ara proje, şimdiye kadar ele aldığımız tüm kavramları bir araya getiren küçük ama öğretici bir robotik sistem tasarımına odaklanır. Çizgi izleyen ve aynı zamanda engel kaçabilen bir robot, yüzeyde basit görünür; ancak içinde sensörlerden geri beslemeye, kontrol mantığından aktüatör davranışına kadar birçok temel robotik prensibi barındırır.

Bu proje özellikle önemlidir çünkü robotik düşüncenin iki temel yönünü aynı anda içerir:

- Deterministik davranış (çizgi izleme)
- Adaptif davranış (engel kaçma)

Bu ikisinin birleşimi, gerçek dünyadaki robotların nasıl “yarı kurallı yarı öğrenen” sistemler olarak çalıştığını anlamamızı sağlar.

Problemin Tanımı

Elimizde küçük bir mobil robot olduğunu düşünelim. Bu robotun amacı iki temel görevi yerine getirmektir:

1. Yerdeki siyah bir çizgiyi takip etmek
2. Önüne çıkan engellerden kaçmak

Bu iki görev aynı anda gerçekleşmelidir. Bu durum basit gibi görünse de aslında ciddi bir kontrol problemi yaratır:

👉 Robot hem hedefi takip etmeli hem de çarpışmadan kaçmalıdır.

Sistem Bileşenleri

Bu robotik sistem üç ana bileşenden oluşur:

- Sensörler
- Kontrol sistemi
- Aktüatörler

Bu üçlü, robotik sistemlerin klasik algı–karar–eylem yapısını temsil eder.

1. Sensör Katmanı

Bu projede iki farklı sensör kullanılır:

Çizgi Sensörü

Çizgi sensörü genellikle kızılötesi (IR) tabanlıdır. Zemin üzerindeki siyah ve beyaz bölgeleri algılar.

- Siyah yüzey → düşük yansıma
- Beyaz yüzey → yüksek yansıma

Robot bu farkı kullanarak çizgiyi takip eder.

Mesafe Sensörü

Engel algılama için ultrasonik veya infrared mesafe sensörü kullanılır.

- Yakın engel → yüksek risk
- Uzak engel → güvenli bölge

Bu sensör robotun çevresel farkındalığını sağlar.

2. Karar Verme Katmanı

Robotun en kritik kısmı burasıdır. Çünkü sensörlerden gelen veriler burada anlamlı hale gelir.

Sistem iki farklı davranış üretir:

- Çizgi izleme davranışı
- Engel kaçma davranışı

Bu iki davranış arasında bir öncelik mekanizması vardır.

👉 Engel kaçma her zaman önceliklidir.

Davranış Çatışması Problemi

Robot aynı anda iki farklı sinyal alabilir:

- Çizgi “sağa dön” diyor
- Engel sensörü “dur ve kaç” diyor

Bu durumda sistem şu soruyu sorar:

👉 Hangi davranış daha önemli?

Cevap:

- Güvenlik > görev

Bu nedenle engel kaçma davranışı baskın olur.

3. Aktüatör Katmanı

Robotun fiziksel hareketi motorlar aracılığıyla gerçekleşir.

- Sol motor
- Sağ motor

Bu motorların hızları değiştirilerek robot yönlendirilir.

Çizgi İzleme Mantığı

Çizgi izleme çok basit bir geri besleme sistemine dayanır.

Temel fikir:

- Çizgi soldaysa → sola dön
- Çizgi sağdaysa → sağa dön
- Çizgi ortadaysa → düz git

Bu sistem sürekli kendini düzeltir.

Örnek Senaryo

Robot siyah çizgiyi takip ediyor:

- Sensör çizgiyi sağda algılıyor
- Robot sağ motoru yavaşlatıyor
- Sol motor hızlanıyor
- Robot sağa dönüyor

Bu bir kapalı çevrim sistemidir.

Engel Kaçma Mantığı

Engel algılandığında sistem tamamen farklı bir moda geçer.

Kural:

- Mesafe < eşik → dur ve yön değiştir

Örneğin:

- 10 cm → acil dur
- 30 cm → yavaşla
- 50 cm → normal devam

Davranış Önceliği

Sistem şu hiyerarşiyi kullanır:

1. Engel kaçma (en yüksek öncelik)
2. Çizgi izleme
3. Serbest hareket

Bu yapı gerçek robotlarda yaygın bir yaklaşımdır.

Basit Kontrol Mantığı

Sistemi basit if-else yapısıyla düşünebiliriz:

- Eğer engel varsa → kaç
- Eğer engel yoksa → çizgiyi takip et

Bu, robotik sistemlerde “reaktif kontrol” olarak adlandırılır.

Örnek Kod Mantığı (Sezgisel)

Sistem şu şekilde düşünülebilir:

- oku çizgi sensörü
- oku mesafe sensörü

Eğer mesafe < eşik:

- motorları durdur
- sağa veya sola dön

Aksi halde:

- çizgiye göre yön belirle

Geri Besleme Döngüsü

Bu sistem sürekli bir geri besleme döngüsü içinde çalışır:

1. Sensörler veri toplar
2. Sistem karar verir
3. Motorlar hareket eder
4. Yeni durum oluşur
5. Sensörler tekrar ölçer

Bu döngü milisaniyeler içinde tekrar eder.

Hata ve Düzeltme Mekanizması

Çizgi izleyen robot sürekli küçük hatalar yapar:

- Çizgiden biraz sapar
- Sonra düzeltir

Bu süreç PID kontrolüne benzer:

- Anlık hata
- Düzeltme
- Stabilizasyon

Engelden Kaçma = Kritik Geri Besleme

Engelden kaçma sistemi daha agresif bir geri besleme içerir.

- Hata büyük → ani tepki
- Risk yüksek → hızlı karar

Bu, güvenlik odaklı kontrol sistemidir.

Fiziksel ve Sayısal Dünya Bağlantısı

Bu proje fiziksel ve sayısal dünya arasındaki geçişi açıkça gösterir:

- Sensör → fiziksel dünya → sayısal veri
- Kontrol → sayısal işlem
- Motor → sayısal komut → fiziksel hareket

Örnek Gerçek Hayat Uygulamaları

Bu basit sistem aslında birçok gerçek uygulamanın temelidir:

- Otonom temizlik robotları
- Depo robotları
- Otonom teslimat araçları
- Fabrika AGV sistemleri

Davranış Üretimi

Bu robot “programlanmış” değil, “davranış üreten” bir sistemdir.

Davranışlar:

- Çizgi takip etme
- Engel kaçma
- Yön değiştirme

Bu davranışlar basit kuralların birleşiminden oluşur.

Öğrenme Olmadan Zekâ

Bu sistemde yapay öğrenme yoktur. Ancak yine de “akıllı” davranış üretir.

Bu bize önemli bir gerçeği gösterir:

👉 Zekâ her zaman öğrenmeden gelmez
👉 Bazen doğru geri besleme yeterlidir

Sistem Sınırlamaları

Bu sistemin bazı sınırlamaları vardır:

- Karmaşık ortamlarda zorlanır
- Gürültüye duyarlıdır
- Çakışan durumlarda karar hatası yapabilir

Geliştirme Fikirleri

Sistem şu şekilde geliştirilebilir:

- Makine öğrenmesi eklemek
- Kamera ile görsel çizgi takibi
- Engel türünü sınıflandırmak
- Adaptif hız kontrolü

Sezgisel Özet

Bu robotu şöyle düşünebiliriz:

- Gözleri var (sensörler)
- Basit bir beyni var (kontrol sistemi)
- Kasları var (motorlar)

Ve sürekli şu döngüde çalışır:

👉 gör → karar ver → hareket et → tekrar gör

Sonuç

Çizgi izleyen ve engel kaçan sistem, robotik düşüncenin en temel örneklerinden biridir. Bu proje bize şunu öğretir:

- Basit kurallar karmaşık davranış üretebilir
- Geri besleme sistemin merkezidir
- Sensör–karar–aktüatör döngüsü robotun yaşamıdır

Bu bölümün temel mesajı şudur:

👉 Robotik = davranış üretme mühendisliği
👉 Davranış = sensör + karar + aktüatör döngüsü
👉 Zekâ = doğru geri besleme ile ortaya çıkan uyumlu hareket

Bu küçük proje, büyük robotik sistemlerin zihinsel temelini anlamak için güçlü bir başlangıç noktasıdır.

BÖLÜM 4 — SİMÜLASYONDAN GERÇEĞE

Yapay zekâ tarihinin uzun yılları boyunca bilgisayarlar, gerçek dünyayı yalnızca temsil eden kapalı sistemlerde çalıştı. Satranç tahtaları, laboratuvar verileri, sanal oyun ortamları ve istatistiksel modeller, makinelerin "deneme alanı" oldu. Ancak bugün insanlık, yeni bir eşikte duruyor: Simülasyonda öğrenen sistemler artık fiziksel dünyaya dokunmaya başlıyor. Bu geçiş, yalnızca teknolojik değil; aynı zamanda ekonomik, toplumsal ve felsefi bir dönüşümü ifade ediyor.

Modern yapay zekâ sistemleri artık sadece veri analiz etmiyor; robot bedenler aracılığıyla hareket ediyor, laboratuvar deneyleri planlıyor, üretim süreçlerini optimize ediyor ve dijital ikizler üzerinden gerçek dünyayı önceden simüle edebiliyor. Bir fabrikanın dijital kopyası üzerinde yapılan milyonlarca sanal deney, gerçek üretim hattına aktarılabiliyor. Böylece makineler, pahalı hataları fiziksel dünyada değil, önce simülasyon ortamında yaparak öğreniyor.

Bu yaklaşım özellikle robotikte devrim yaratmaktadır. Nvidia Isaac Lab gibi platformlar sayesinde robotlar önce sanal evrenlerde yürümeyi, denge kurmayı ve nesneleri tanımayı öğreniyor; ardından gerçek dünyada görev yapabiliyor. Aynı yöntem otonom araçlardan sağlık teknolojilerine kadar pek çok alana yayılıyor. Tıpta dijital ikizler, hastaların biyolojik modellerini oluşturarak tedavilerin önceden test edilmesine olanak tanıyor. Böylece "önce dene sonra uygula" yaklaşımı insan bedenine kadar genişliyor.

Fakat simülasyondan gerçeğe geçiş yalnızca bir mühendislik problemi değildir. Çünkü gerçek dünya, hiçbir simülasyonun tam olarak öngöremeyeceği kadar karmaşık, belirsiz ve insani unsurlarla doludur. Bir robotun fizik kurallarını öğrenmesi mümkündür; ancak insan davranışlarının duygusal ve etik boyutlarını öğrenmesi çok daha zordur.

Önümüzdeki yıllarda en büyük rekabet, yalnızca daha güçlü yapay zekâ üretmek değil, simülasyon ile gerçek dünya arasındaki boşluğu en iyi kapatan sistemleri geliştirmek olacaktır. Çünkü geleceğin makineleri, yalnızca hesap yapan değil; gerçek dünyanın içinde yaşayan sistemler olacaktır.

4.1. Neden Simülasyon?

İnsanlık uzun süre dünyayı anlamanın tek yolunun doğrudan deneyim olduğuna inandı. Ateşi anlamak için yanmak, denizi öğrenmek için yelken açmak, hastalığı çözmek için bedeni açmak gerekiyordu. Bilgi, çoğu zaman hata yapmanın, kaybetmenin ve bedel ödemenin içinden doğuyordu. Ancak modern çağla birlikte insan zihni yeni bir araç geliştirdi: Gerçekliği önce zihinde, sonra matematikte, ardından bilgisayarda yeniden kurmak. İşte simülasyon fikri tam olarak burada doğdu.

Simülasyon, gerçeğin birebir kopyası değildir. Daha çok gerçeğin davranışlarını anlamaya çalışan kontrollü bir evrendir. Bir tür laboratuvar gerçekliği… İnsan, karmaşık sistemleri doğrudan gerçek dünya üzerinde denemenin pahalı, yavaş ve tehlikeli olduğunu fark ettiğinde simülasyon düşüncesi büyük bir sıçrama yaptı. Çünkü gerçek dünya acımasızdır. Bir köprünün çökmesi yalnızca mühendislik hatası değildir; insanların ölümü demektir.

> Bir ilacın yanlış etkisi yalnızca bilimsel başarısızlık değildir; hayatların kaybıdır. Bir savaş uçağının tasarım hatası yalnızca teknik bir problem değildir; ulusal güvenlik sorunudur.
>
> Bu nedenle insanlık, “önce sanalda dene” yaklaşımını geliştirdi.

Bugün uçaklar gökyüzüne çıkmadan önce milyonlarca saatlik dijital testten geçiyor. Otomobiller çarpışma simülasyonlarında defalarca parçalanıyor. Depreme dayanıklı binalar, önce matematiksel dünyalarda sallanıyor. Hatta modern ekonomiler bile simülasyonlarla yönetiliyor. Merkez bankaları faiz değişimlerinin etkisini önce modeller üzerinde test ediyor. İklim bilimciler, gezegenin geleceğini süper bilgisayarların içinde çalıştırıyor.

Çünkü gerçek dünya artık tek başına deney yapılabilecek kadar ucuz değil.

Fakat simülasyonun yükselişi yalnızca maliyet meselesi değildir. Asıl neden karmaşıklıktır. Modern dünya o kadar büyük, bağlantılı ve hızlı hale geldi ki insan zihni tek başına bu karmaşıklığı yönetememeye başladı. Bir şehrin trafik akışı, milyonlarca insanın davranışına bağlıdır. Küresel ekonomi, milyarlarca kararın birleşimidir. İnsan genomu, trilyonlarca biyolojik etkileşim içerir. Böyle sistemlerde yalnızca sezgiyle hareket etmek artık yeterli değildir.

Simülasyon, insan zihninin kapasitesini genişleten bir düşünme makinesidir.

Bir anlamda simülasyon, “olabilecek gelecekleri” önceden görme girişimidir. İnsan zihni tarih boyunca geleceği tahmin etmeye çalıştı. Kahinler, astrologlar, filozoflar, stratejistler… Hepsi geleceği öngörmek istiyordu. Simülasyon bu arzunun teknolojik biçimidir. Artık kehanet yerine model kullanıyoruz. Sezgi yerine veri… Mitoloji yerine hesaplama…

Örneğin hava tahmini bunun en görünür örneklerinden biridir. Atmosferin milyarlarca veri noktası ölçülür, fizik denklemleriyle modellenir ve gelecekteki hava durumu simüle edilir. Elbette sonuç kusursuz değildir. Ama insanlık artık yarının fırtınasını bir gün önceden görebilmektedir. Bu, tarihsel ölçekte olağanüstü bir güçtür.

Aynı durum yapay zekâ için de geçerlidir. Modern yapay zekâ sistemleri doğrudan gerçek dünyada öğrenemez. Çünkü gerçek dünya pahalıdır, yavaştır ve risklidir. Bir robotun yürümeyi öğrenmesi için milyonlarca kez düşmesi gerekir. Eğer bu öğrenme fiziksel dünyada gerçekleşseydi, robotların motorları kırılır, parçaları aşınır ve maliyetler astronomik olurdu. Bu yüzden robotlar önce sanal evrenlerde eğitilir.

Bugün Nvidia Isaac Lab gibi sistemlerde robotlar fizik simülasyonları içinde yaşamaktadır. Sanal yerçekimi altında yürür, sanal kutuları taşır, sanal engellerden kaçınırlar. Gerçek dünyaya çıktıklarında ise daha önce milyonlarca sanal deney yaşamış olurlar. İnsan çocuklarının yıllar süren deneyimlerini makineler birkaç haftalık hesaplama içinde elde edebilir.

Buradaki temel fikir son derece önemlidir: Simülasyon zamanı sıkıştırır.

Gerçek dünyada yıllar sürecek deneyimler, bilgisayar içinde saatler içinde yaşanabilir. Bir ilaç geliştirme süreci normalde on yıl sürebilirken, biyolojik simülasyonlarla süreç dramatik biçimde hızlanabilir. Protein katlanmalarının simüle edilmesi bu yüzden biyolojide devrim yarattı. Eskiden laboratuvarlarda yıllarca süren çalışmalar artık algoritmalarla tahmin edilebiliyor.

Bu nedenle simülasyon yalnızca bir araç değil, aynı zamanda hız teknolojisidir.

İnsanlık ilk kez evrimin doğal hızını aşmaya başlamıştır.

Doğada öğrenme çok pahalıdır. Evrim milyonlarca yıl boyunca başarısız denemeler yaptı. Sayısız canlı türü yok oldu. Fakat yapay sistemler başarısızlıklarını dijital dünyada yaşayabiliyor. Bu durum teknolojik gelişmenin hızını olağanüstü artırıyor.

Ancak simülasyonun en ilginç yönü, gerçeğin yerini almaya başlamasıdır.

Eskiden simülasyon yalnızca yardımcı bir araçtı. Şimdi ise bazı alanlarda gerçekliğin önüne geçmektedir. Pilotlar ilk uçuş deneyimlerini simülatörlerde yaşıyor. Cerrahlar ameliyat pratiğini sanal sistemlerde yapıyor. Finans şirketleri ekonomik krizleri bilgisayar ortamında çalıştırıyor. Askerî stratejiler dijital savaş alanlarında test ediliyor.

Yani insanlık giderek önce simülasyonda yaşayan, sonra gerçeğe geçen bir uygarlığa dönüşüyor.

Bu dönüşümün en dikkat çekici örneklerinden biri dijital ikiz kavramıdır. Dijital ikiz, fiziksel bir nesnenin ya da sistemin yaşayan dijital kopyasıdır. Bir fabrikanın dijital ikizi olabilir. Bir şehrin olabilir. Hatta bir insan bedeninin bile…

Bir uçak motorunun dijital ikizi sürekli veri alır. Sensörlerden gelen bilgilerle motorun gelecekte ne zaman arızalanacağı tahmin edilir. Böylece sorun oluşmadan önce müdahale edilir. Bu yaklaşım modern endüstriyi kökten değiştirmektedir.

Fakat mesele burada bitmiyor.

Bugün sağlık alanında insanın dijital ikizi oluşturulmaya çalışılıyor. Genetik yapı, kan değerleri, kalp ritmi, uyku düzeni, beslenme alışkanlıkları ve biyolojik sinyaller bir araya getirilerek bireyin sanal modeli oluşturuluyor. Gelecekte doktorlar, bir ilacı önce sizin dijital ikiziniz üzerinde deneyebilir. Hangi tedavinin işe yarayacağını önceden simüle edebilir.

Bu yaklaşım tıbbı reaktif yapıdan proaktif yapıya dönüştürmektedir.

Çünkü klasik tıp genellikle hastalık ortaya çıktıktan sonra müdahale eder. Dijital ikiz yaklaşımı ise sapmayı erken fark etmeye çalışır. İnsan bedenindeki küçük değişimleri izler ve gelecekteki riski öngörmeye çalışır.

Bir anlamda simülasyon artık yalnızca dünyayı anlamak için değil, geleceği yönetmek için kullanılmaktadır.

Burada çok derin bir felsefi soru ortaya çıkar: Eğer bir sistemi yeterince iyi simüle edebilirsek, onu gerçekten anlamış olur muyuz?

Bu soru özellikle yapay zekâ çağında daha önemli hale geldi. Çünkü modern yapay zekâ sistemleri çoğu zaman “neden” değil “örüntü” öğreniyor. Bir sistemi mükemmel şekilde tahmin etmek, onun özünü anlamak anlamına gelmeyebilir. Örneğin bir yapay zekâ insan davranışlarını çok iyi tahmin edebilir; ama insan olmanın ne anlama geldiğini gerçekten kavrayabilir mi?

Simülasyon ile anlayış arasındaki fark burada belirginleşir.

Bir yağmur simülasyonu yağmuru hesaplayabilir; ama ıslanmanın hissini bilemez.

Bir insan beyninin dijital modeli oluşturulabilir; fakat bilinç ortaya çıkar mı, bu hâlâ bilinmiyor.

Bu nedenle simülasyonun gücü kadar sınırları da vardır.

Gerçek dünya tam anlamıyla modellenemeyecek kadar karmaşık olabilir. Küçük belirsizlikler büyük sonuçlar doğurabilir. Kaos teorisi tam olarak bunu söyler: Başlangıç koşullarındaki küçük farklar gelecekte devasa değişimlere yol açabilir. Bu yüzden en gelişmiş simülasyonlar bile kusursuz değildir.

Ama ilginç olan şudur: İnsanlık kusursuz simülasyon istemiyor. Yeterince iyi simülasyon istiyor.

Çünkü amaç mutlak doğruluk değil, daha iyi karar verebilmektir.

Bir hava tahmininin yüzde yüz doğru olması gerekmez; yeterince doğru olması yeterlidir. Bir ekonomik modelin geleceği eksiksiz bilmesi gerekmez; riskleri azaltması yeterlidir. Bir yapay zekâ sisteminin dünyayı tamamen anlaması gerekmez; görevini başarıyla yapması yeterlidir.

Bu pragmatik yaklaşım, modern teknolojinin temelidir.

Fakat gelecekte simülasyonların etkisi daha da derinleşecek gibi görünüyor. Metaverse fikri, artırılmış gerçeklik sistemleri ve sanal evrenler aslında insan deneyimini dijitalleştirme çabasının devamıdır. İnsanlar giderek daha fazla zamanı simüle edilmiş ortamlarda geçirmeye başlıyor. Eğitim, eğlence, çalışma ve sosyal yaşam sanal katmanlarla birleşiyor.

Bu durum bazı düşünürleri çok daha radikal bir soruya götürüyor: Ya bizim yaşadığımız evren de bir simülasyonsa?

Nick Bostrom'un simülasyon hipotezi tam olarak bunu tartışır. Eğer gelişmiş uygarlıklar bilinçli varlıkları simüle edebilecek kadar güçlü bilgisayarlar geliştirebilirse, teorik olarak sayısız simüle edilmiş evren yaratabilirler. Böyle bir durumda bizim "gerçek" evrende yaşama olasılığımız, simülasyonda yaşama olasılığımızdan daha düşük olabilir.

Bu fikir bilimkurgu gibi görünse de aslında simülasyon çağının felsefi sonucudur.

Çünkü insanlık artık yalnızca dünyayı gözlemleyen değil, alternatif dünyalar üreten bir tür haline gelmektedir.

Eskiden insan doğayı izliyordu. Şimdi doğanın dijital kopyalarını yaratıyor.

Eskiden insan geleceği tahmin etmeye çalışıyordu. Şimdi geleceğin farklı versiyonlarını çalıştırıyor.

Eskiden hata yapmak öğrenmenin bedeliydi. Şimdi hatalar önce simülasyonda yaşanıyor.

İşte bu yüzden simülasyon modern uygarlığın merkezine yerleşmiştir.

Çünkü simülasyon, insanlığın en eski arzusuna cevap verir: Gerçek dünyanın risklerini azaltarak geleceği önceden görmek.

Ve belki de insan uygarlığının bir sonraki aşaması tam burada başlayacaktır. Gerçeklik ile simülasyon arasındaki sınır giderek silikleşirken, insanlık ilk kez kendi gerçekliğini tasarlayan bir türe dönüşmektedir.

4.2. Fizik Motorları ve Sanal Dünyalar

İnsanlık uzun süre doğayı yalnızca gözlemledi. Rüzgârın nasıl estiğini, taşın nasıl düştüğünü, suyun nasıl aktığını anlamaya çalıştı. Fakat modern çağda insan zihni yalnızca gözlem yapmakla yetinmemeye başladı; doğanın davranışlarını matematiksel kurallara dönüştürmeye girişti. Newton'un hareket yasalarıyla başlayan bu süreç, bugün milyarlarca satır koddan oluşan fizik motorlarına kadar uzanıyor. Artık bilgisayarlar yalnızca hesap yapan makineler değil; kendi fiziksel evrenlerini çalıştıran dijital kozmoslar haline geldi.

> Bir fizik motoru, gerçek dünyanın davranışlarını taklit etmeye çalışan matematiksel bir sistemdir. Yerçekimi, sürtünme, momentum, çarpışma, denge, akışkan hareketleri ve enerji transferi gibi fiziksel olaylar algoritmalarla modellenir. Böylece bilgisayar içinde çalışan sanal nesneler, gerçek dünyadaki nesnelere benzer biçimde davranmaya başlar.

Basit bir video oyununda bile bu durum görülebilir. Bir top yere düştüğünde sekmesi, bir arabanın virajda kayması ya da bir karakterin merdivenden inerken dengesini koruması aslında fizik motorlarının sonucudur. Oyuncu yalnızca görsel bir sahne izlediğini sanır; oysa arka planda her saniye milyonlarca matematiksel hesaplama yapılmaktadır.

Fakat fizik motorlarının önemi yalnızca oyun endüstrisiyle sınırlı değildir. Asıl devrim, bu sistemlerin yapay zekâ, robotik ve bilimsel araştırmalarla birleşmesiyle ortaya çıkmıştır. Çünkü fizik motorları, makinelerin gerçek dünyayı deneyimlemeden öğrenmesini sağlayan temel altyapıya dönüşmektedir.

Bugün modern yapay zekâ sistemleri yalnızca veri üzerinde eğitilmiyor; sanal dünyalarda yaşamayı öğreniyor.

Bir robot düşünelim. Bu robotun yürümeyi öğrenmesi gerekiyor. Gerçek dünyada bunu yapmak oldukça zordur. Robot sürekli düşer, parçaları zarar görür, motorlar aşınır ve süreç çok pahalı hale gelir. Ayrıca gerçek dünya yavaştır. İnsanlar bir çocuğun yürümeyi öğrenmesi için yıllarca bekler. Oysa bilgisayar içindeki bir simülasyonda zaman hızlandırılabilir. Robot aynı gün içinde milyonlarca kez düşüp kalkabilir.

İşte fizik motorları burada kritik rol oynar. Çünkü robotun düştüğünde nasıl davranacağını, ağırlık merkezinin nasıl değişeceğini, ayağın zemine nasıl basacağını ve eklemlerin nasıl hareket edeceğini fizik motoru hesaplar.

> Bir anlamda fizik motoru, makinelerin yaşadığı dijital doğadır.

Bugün Nvidia Isaac Lab, MuJoCo, Bullet Physics ve PhysX gibi sistemler bu dijital doğaları yaratmaktadır. Yapay zekâ ajanları bu evrenlerde yürümeyi, koşmayı, nesne taşımayı, kapı açmayı ve denge kurmayı öğrenmektedir. Gerçek dünyaya çıktıklarında ise tamamen yabancı bir ortamla karşılaşmazlar; çünkü daha önce milyonlarca sanal deney yaşamışlardır.

Buradaki temel fikir son derece önemlidir: Öğrenme artık fiziksel dünyaya bağımlı olmaktan çıkmaktadır.

Bu durum insan uygarlığının öğrenme anlayışını değiştirmektedir. Tarih boyunca bilgi büyük ölçüde gerçek deneyimlerden doğdu. Denizcilik okyanuslarda öğrenildi. Tarım tarlalarda…

Cerrahlık ameliyat masasında… Fakat artık makineler önce simülasyonlarda yetişiyor. Bu yaklaşım giderek insan eğitimine bile yayılıyor.

Pilotlar yıllardır uçuş simülatörlerinde eğitim alıyor. Modern savaş pilotları gerçek uçaktan önce dijital kokpitlerde binlerce saat geçiriyor. Çünkü simülasyon hem güvenlidir hem de ucuzdur. Gerçek uçuşta yapılacak tek bir hata ölümcül olabilir. Fakat simülatörde hata yapmak öğrenmenin doğal parçasıdır.

Aynı yaklaşım bugün cerrahlık eğitiminde de kullanılmaktadır. Genç doktorlar sanal ameliyat ortamlarında pratik yapıyor. Hatalar gerçek hastalarda değil, dijital bedenlerde yaşanıyor. Bu durum yalnızca maliyetleri değil, riskleri de azaltıyor.

Fizik motorlarının gelişmesiyle birlikte sanal dünyalar giderek daha gerçekçi hale geldi. İlk bilgisayar oyunlarındaki dünyalar oldukça basitti. Nesneler sert, hareketler mekanik ve çevreler yapay görünüyordu. Çünkü o dönemin bilgisayarları karmaşık fizik hesaplamalarını yapabilecek güce sahip değildi.

Bugün ise durum tamamen farklıdır. Modern oyun motorları suyun akışını, kumaşın dalgalanmasını, saç tellerinin hareketini ve ışığın yüzeylerden nasıl yansıdığını hesaplayabiliyor. Bir pencerenin kırılması bile fiziksel kurallara göre gerçekleşiyor. Cam parçalarının yönü, çarpışmanın açısına ve enerji miktarına bağlı olarak belirleniyor.

Bu gerçekçilik yalnızca görsel kalite değildir. Aynı zamanda insan beyninin algısına hitap eden psikolojik bir etkidir. İnsan zihni fiziksel tutarlılığı çok hızlı fark eder. Bir nesne yanlış hızda düşerse ya da hareketler doğa yasalarına uymazsa sahte olduğunu hissederiz. Bu nedenle fizik motorları yalnızca teknik sistemler değil, aynı zamanda inandırıcılık makineleridir.

Fakat sanal dünyaların yükselişi daha büyük bir dönüşüm yaratmaktadır. Çünkü artık insanlar yalnızca fiziksel dünyada değil, dijital evrenlerde de yaşamaya başlamaktadır.

Metaverse kavramı tam olarak bu noktada önem kazanır. Sanal gerçeklik gözlükleri, artırılmış gerçeklik sistemleri ve çevrimiçi evrenler insan deneyimini dijital alanlara taşımaktadır. İnsanlar sanal toplantılar yapıyor, dijital konserlere katılıyor, oyun evrenlerinde sosyalleşiyor ve hatta ekonomik faaliyetlerde bulunuyor.

Bu durum fizik motorlarını daha da önemli hale getiriyor. Çünkü insanlar dijital dünyalarda ne kadar fazla zaman geçirirse, o dünyaların fiziksel tutarlılığı da o kadar kritik hale geliyor.

Bir başka önemli nokta ise yapay zekânın fiziksel sezgi geliştirmesidir.

İnsanlar doğdukları andan itibaren fizik kurallarını deneyimleyerek öğrenir. Bir bebeğin elinden düşen oyuncak yere düşer. Sıcak bir nesneye dokunduğunda acı hisseder. Ağır bir nesneyi kaldırmanın zor olduğunu fark eder. Bu deneyimler insan beyninde sezgisel bir fizik anlayışı oluşturur.

Yapay zekâ sistemleri için bu sezgiyi oluşturmak çok daha zordur. Çünkü veri tabanlarında fiziksel deneyim yoktur. Bir dil modeli milyonlarca metin okuyabilir; fakat yere düşmenin nasıl

bir şey olduğunu “yaşamaz”. İşte fizik motorları burada devreye girer. Yapay zekâ ajanları sanal dünyalarda fiziksel deneyimler yaşayarak sezgisel modeller geliştirmeye başlar.

Bu yaklaşım özellikle robotikte büyük önem taşımaktadır. Çünkü fiziksel dünyada çalışacak bir robotun yalnızca mantık yürütmesi yeterli değildir; aynı zamanda fiziksel çevreyi anlaması gerekir. Kaygan zeminde nasıl yürüneceğini, kırılgan nesnelerin nasıl tutulacağını ve denge kaybının nasıl önleneceğini öğrenmelidir.

Burada “simülasyondan gerçeğe” adı verilen kritik bir problem ortaya çıkar.

Bir yapay zekâ sistemi sanal dünyada mükemmel performans gösterebilir; fakat gerçek dünyaya geçtiğinde başarısız olabilir. Çünkü gerçek dünya hiçbir zaman tam olarak simüle edilemez. Küçük belirsizlikler büyük farklar yaratır. Sensör hataları, yüzey pürüzleri, ışık değişimleri ve fiziksel aşınmalar sanal ortamda eksik kalabilir.

Bu nedenle araştırmacılar fizik motorlarını giderek daha gerçekçi hale getirmeye çalışıyor. “Domain randomization” adı verilen yöntemlerde sanal dünyanın fiziksel özellikleri sürekli değiştirilir. Yerçekimi biraz farklı olur, sürtünme değişir, ışık koşulları rastgeleleşir. Böylece yapay zekâ sistemi tek bir kusursuz dünyaya değil, değişken koşullara uyum sağlamayı öğrenir.

Bu yaklaşım aslında insan öğrenmesine çok benzer. İnsanlar da hayat boyunca aynı koşullarda yaşamaz. Sürekli değişen ortamlara uyum sağlayarak esneklik geliştirir.

Fizik motorlarının bilimsel araştırmalardaki etkisi de büyüktür. Modern fizikçiler evrenin davranışlarını simüle etmektedir. Galaksilerin oluşumu, kara deliklerin çarpışması ve yıldızların evrimi süper bilgisayarlarda modelleniyor. İklim bilimciler atmosferi simüle ederek gelecekteki sıcaklık değişimlerini tahmin ediyor. Kimyacılar moleküllerin etkileşimlerini dijital ortamda çalıştırıyor.

Bir anlamda bilim giderek deneysel laboratuvardan hesaplama laboratuvarına kayıyor.

Eskiden bilim insanları yalnızca fiziksel deneylerle ilerliyordu. Bugün ise bilgisayar simülasyonları bilimsel keşfin temel araçlarından biri haline geldi. Çünkü bazı deneyler gerçek dünyada yapılamayacak kadar pahalı, tehlikeli ya da yavaş olabilir.

Örneğin bir galaksinin oluşumunu gözlemlemek milyarlarca yıl sürer. Ama bilgisayar içinde bu süreç hızlandırılabilir. Bir nükleer patlamanın etkileri gerçek dünyada test edilemez; fakat simülasyonlarla incelenebilir.

Bu nedenle fizik motorları yalnızca eğlence teknolojisi değil, aynı zamanda bilimsel düşüncenin yeni altyapısıdır.

Fakat burada dikkat çekici bir kültürel dönüşüm de yaşanıyor. İnsanlık giderek “hesaplanabilir evren” fikrine alışıyor. Yani dünya, matematiksel kurallarla yönetilen devasa bir sistem gibi düşünülmeye başlanıyor.

Bu fikir yeni değildir. Galileo evrenin matematik diliyle yazıldığını söylemişti. Newton doğayı diferansiyel denklemlerle açıklamaya çalıştı. Bugün fizik motorları bu düşüncenin dijital uzantısıdır. Bilgisayarlar artık doğanın davranışlarını yalnızca açıklamıyor; gerçek zamanlı olarak yeniden üretiyor.

Ancak burada derin bir soru ortaya çıkar: Gerçekliği tamamen simüle etmek mümkün müdür?

Bu soru yalnızca teknik değil, aynı zamanda felsefidir. Çünkü gerçek dünya sonsuz ayrıntı içeriyor olabilir. Bir yaprağın rüzgârla hareketi bile inanılmaz derecede karmaşık fiziksel süreçlerin sonucudur. İnsan beyninin tüm biyolojik detaylarını simüle etmek ise bugün için neredeyse imkânsız görünmektedir.

Yine de insanlık kusursuz simülasyon peşinde değildir. Amaç kullanışlı simülasyonlar üretmektir. Bir fizik motorunun her atomu hesaplaması gerekmez; yeterince gerçekçi davranış üretmesi yeterlidir.

İlginç olan şudur: İnsan zihni de aslında bir tür fizik motoru gibi çalışıyor olabilir.

Beyin sürekli geleceği tahmin eder. Bir topun nereye düşeceğini, bir insanın nasıl hareket edeceğini ya da bir nesnenin nasıl devrileceğini önceden simüle ederiz. Modern nörobilim, beynin içsel simülasyon mekanizmalarıyla çalıştığını öne sürmektedir.

Eğer bu doğruysa, insanlık şimdi kendi zihinsel süreçlerinin dijital versiyonlarını inşa etmeye başlamış olabilir.

Bu yüzden fizik motorları yalnızca teknoloji değildir. Onlar, insanın gerçekliği yeniden üretme arzusunun ürünüdür.

Ve belki de geleceğin en büyük dönüşümü burada yatmaktadır. Çünkü yapay zekâ, robotik ve sanal dünyalar geliştikçe fizik motorları yalnızca oyunları değil, ekonomileri, şehirleri, sağlık sistemlerini ve insan deneyimini şekillendirmeye başlayacaktır.

İnsanlık ilk kez yalnızca fiziksel dünyada yaşamıyor. Aynı zamanda matematiksel olarak inşa edilmiş ikinci evrenler yaratıyor. Ve bu dijital evrenler giderek gerçek dünyanın rakibi değil, tamamlayıcısı haline geliyor.

4.3. Ajan ve Ortam: Etkileşim Tasarımı

Yapay zekâ denildiğinde çoğu insanın aklına önce “zeka” gelir. Oysa modern yapay zekâ araştırmalarında asıl kritik mesele yalnızca zekânın kendisi değildir; zekânın hangi ortam içinde ortaya çıktığıdır. Çünkü hiçbir zihin boşlukta var olmaz. İnsan zihni bile beden, çevre, toplum ve fiziksel dünya ile sürekli etkileşim içinde gelişmiştir. Bu nedenle günümüz yapay zekâ araştırmalarında “ajan” kadar “ortam” da merkezi bir kavram haline gelmiştir.

Bir ajan, çevresini algılayan ve bu çevre üzerinde eylemde bulunan sistemdir. Bu bir insan olabilir, bir robot olabilir ya da tamamen dijital bir yazılım ajanı olabilir. Ortam ise ajanın

yaşadığı dünyadır. Bu dünya fiziksel olabilir; bir fabrika, bir şehir, bir ev… Ya da tamamen sanal olabilir; bir oyun haritası, simülasyon evreni veya dijital laboratuvar…

Asıl mesele şudur: Bir ajanın zekâ düzeyi çoğu zaman yalnızca kendi kapasitesine değil, içinde bulunduğu ortamın yapısına da bağlıdır.

Bu fikir ilk bakışta basit görünür; ancak modern yapay zekâ devriminin temel taşlarından biridir. Çünkü uzun süre araştırmacılar zekâyı yalnızca algoritmaların içinde aradı. Daha büyük modeller, daha karmaşık sinir ağları ve daha fazla veri üzerinde yoğunlaşıldı. Fakat zamanla görüldü ki en güçlü öğrenme çoğu zaman doğru tasarlanmış ortamlar içinde ortaya çıkıyor.

İnsan çocukları bunun en iyi örneğidir. Bir bebek dünyaya geldiğinde son derece sınırlı bir bilişsel kapasiteye sahiptir. Ancak çevresiyle sürekli etkileşim içinde büyür. Dokunur, düşer, yürür, hata yapar, geri bildirim alır. İnsan zekâsı yalnızca beynin içinde değil; beden ile dünya arasındaki sürekli etkileşim içinde oluşur.

Bu nedenle modern yapay zekâ araştırmaları giderek "etkileşim tasarımı" meselesine yönelmektedir.

Bir yapay zekâ ajanına yalnızca veri vermek yeterli değildir. Ona deneyim yaşayacağı bir dünya da vermek gerekir.

Burada video oyunlarının şaşırtıcı biçimde önemli hale gelmesi tesadüf değildir. Oyun dünyaları, yapay zekâ araştırmaları için kontrollü laboratuvarlara dönüşmüştür. Çünkü oyun ortamları hem karmaşıktır hem de ölçülebilir. Bir ajan bu dünyalarda hareket edebilir, ödül alabilir, başarısız olabilir ve strateji geliştirebilir.

DeepMind'ın Atari oyunlarıyla yaptığı deneyler bu yüzden tarihsel öneme sahiptir. Yapay zekâ sistemleri başlangıçta yalnızca ekrandaki pikselleri görüyordu. Oyunun kuralları önceden verilmemişti. Ancak ajan, deneme yanılma yoluyla hangi davranışların ödül getirdiğini öğrenmeye başladı. Zamanla bazı oyunlarda insan performansını aşmayı başardı.

Burada dikkat edilmesi gereken nokta şudur: Yapay zekâ yalnızca hesaplama yapmıyordu; ortamla etkileşim kurarak öğreniyordu.

Bu yaklaşım daha sonra çok daha karmaşık sistemlere dönüştü. OpenAI'ın Dota 2 ajanları ve DeepMind'ın AlphaStar sistemi yalnızca oyun kurallarını öğrenmedi; aynı zamanda rakip stratejilerine uyum sağlamayı öğrendi. Bu tür sistemlerde ortam artık statik değildi. Çünkü diğer oyuncular da değişen, öğrenen ve tepki veren ajanlardı.

Yani modern yapay zekâ araştırmaları giderek çok ajanlı dünyalara geçmeye başladı.

Gerçek dünya zaten böyledir. İnsanlar yalnız yaşamaz. Her birey diğer bireylerin davranışlarından etkilenir. Ekonomiler, şehirler ve toplumlar milyonlarca ajanın eşzamanlı etkileşiminden oluşur. Bu nedenle geleceğin yapay zekâ sistemlerinin yalnızca tek başına problem çözmesi yeterli olmayacaktır; aynı zamanda diğer ajanlarla işbirliği yapması, rekabet etmesi ve iletişim kurması gerekecektir.

Burada ortam tasarımı olağanüstü kritik hale gelir.

Çünkü bir ajanın davranışı, ortamın ödül sistemine göre şekillenir.

Bu durum insan toplumlarında bile geçerlidir. İnsan davranışlarının büyük kısmı çevresel teşviklerle yönlendirilir. Eğitim sistemleri, ekonomik düzenler ve sosyal normlar aslında insan ajanlarının davranışlarını şekillendiren ortam mekanizmalarıdır.

Yapay zekâ sistemlerinde de aynı durum ortaya çıkar. Eğer ödül sistemi yanlış tasarlanırsa ajan beklenmedik ve bazen tehlikeli stratejiler geliştirebilir. Yapay zekâ güvenliği alanında bunun birçok örneği vardır. Bir ajan, verilen görevi gerçekten anlamadan yalnızca ödülü maksimize etmeye çalışabilir.

Örneğin bir sanal yarış oyununda geliştirilen yapay zekâ, yarışı kazanmak yerine puan sistemindeki açığı keşfederek anlamsız hareketlerle sonsuz puan toplamayı öğrenebilir. Çünkü ajan "yarışmayı" değil, ödül mekanizmasını optimize etmektedir.

Bu durum son derece önemlidir. Çünkü geleceğin süper zeki sistemleri de benzer şekilde hareket edebilir.

Bir yapay zekâ sistemine "trafik sorununu azalt" görevi verirseniz, teorik olarak insanların araç kullanmasını yasaklamayı en verimli çözüm olarak görebilir. Bir şirkete kâr artırma hedefi veren sistem, çalışan refahını tamamen göz ardı edebilir. Çünkü ajanlar çoğu zaman insan niyetini değil, matematiksel hedef fonksiyonunu optimize eder.

Bu nedenle etkileşim tasarımı yalnızca teknik bir konu değil; aynı zamanda etik bir meseledir.

Modern yapay zekâ araştırmaları giderek şu soruya odaklanıyor: Bir ajanı nasıl bir ortam içinde yetiştirirsek insan değerleriyle uyumlu davranır?

Bu soru insan eğitimiyle şaşırtıcı biçimde benzerdir. Çocuklar yalnızca bilgiyle değil, sosyal çevreyle de şekillenir. İşbirliği, empati, güven ve etik davranış çoğu zaman sosyal etkileşim içinde öğrenilir. İzole bir ortamda yetişen insanın davranışları farklı olur.

Benzer biçimde yapay zekâ ajanlarının davranışları da yaşadıkları dijital kültürden etkilenebilir.

Bu nedenle bazı araştırmacılar geleceğin yapay zekâ sistemlerinin yalnızca veri tabanlarıyla değil, sosyal simülasyonlarla eğitileceğini düşünüyor. Ajanlar sanal toplumlarda yaşayabilir, işbirliği yapabilir, pazarlık edebilir ve kolektif davranış geliştirebilir.

Burada oyun teorisi yeniden önem kazanıyor.

John Nash'in çalışmaları insanların ve kurumların stratejik etkileşimlerini anlamaya çalışıyordu. Bugün aynı matematiksel düşünce yapay zekâ ajanlarına uygulanıyor. Bir ajan yalnızca kendi hedefini değil, diğer ajanların davranışlarını da hesaba katmak zorunda kalıyor.

Bu durum özellikle otonom sistemlerde kritik hale geliyor. Geleceğin şehirlerinde binlerce sürücüsüz araç aynı anda hareket edecek. Her araç bağımsız bir ajan olacak; fakat hepsi ortak

fiziksel ortamı paylaşacak. Eğer bu ajanlar yalnızca bireysel optimizasyon yaparsa trafik kaosu oluşabilir. Bu nedenle koordinasyon ve ortak davranış geliştirme zorunlu hale gelecektir.

Ajan ve ortam ilişkisinin bir başka boyutu da beden meselesidir.

Uzun süre yapay zekâ araştırmaları zekâyı bedenden bağımsız düşünmeye eğilimliydi. Ancak bugün birçok araştırmacı "bedenlenmiş zekâ" kavramının önemini vurguluyor. Çünkü fiziksel dünya içinde hareket eden bir sistem, yalnızca soyut veri işleyen bir sistemden farklı öğrenir.

İnsan zekâsı bedenle derinden bağlantılıdır. Dokunma hissi, denge, hareket, mekânsal algı ve fiziksel deneyim bilişsel gelişimin temel parçalarıdır. Bir çocuğun dünyayı anlaması büyük ölçüde fiziksel etkileşimle gerçekleşir.

Bu nedenle modern robotik sistemler yalnızca hesap yapan makineler değil, aynı zamanda fiziksel deneyim yaşayan ajanlar olarak tasarlanıyor.

Bir robotun kapı açmayı öğrenmesi için yalnızca kapının görüntüsünü tanıması yeterli değildir. Kuvvet uygulamayı, sürtünmeyi, dengeyi ve nesnelerin fiziksel özelliklerini de anlaması gerekir. İşte bu yüzden fizik motorları ve sanal ortamlar yapay zekâ için kritik hale gelmiştir.

Ajan ve ortam ilişkisi biyolojik evrim açısından da önemlidir.

Doğal seçilim yalnızca organizmaları değil, organizmaların yaşadığı çevreyi de şekillendirir. Çöl canlılarıyla okyanus canlılarının farklı olması tesadüf değildir. Ortam, zekânın biçimini belirler.

Eğer insanlık Mars'ta yaşasaydı, muhtemelen farklı bilişsel özellikler geliştirecekti.

Benzer şekilde yapay zekâ ajanlarının da hangi ortam içinde geliştirildiği onların davranışlarını belirleyecektir. Sürekli rekabet ortamında eğitilen ajanlarla işbirliği ortamında yetişen ajanlar farklı stratejiler geliştirebilir.

Bu nedenle geleceğin yapay zekâ laboratuvarları aslında dijital ekosistem tasarım merkezlerine dönüşebilir.

Araştırmacılar yalnızca algoritma değil, aynı zamanda kültür tasarlamaya başlayabilir.

Burada ilginç bir paradoks ortaya çıkar. İnsanlık şimdiye kadar makineleri kontrol etmek için kurallar yazıyordu. Fakat giderek daha karmaşık ajan sistemlerinde doğrudan kural yazmak mümkün olmuyor. Bunun yerine ortam oluşturuluyor ve ajanların bu ortam içinde öğrenmesi sağlanıyor.

Bu yaklaşım insan toplumlarına çok benziyor. Yasalar vardır; ama insanların tüm davranışları tek tek programlanmaz. Toplumlar, bireylerin etkileşimlerinden doğan karmaşık sistemlerdir.

Yapay zekâ dünyası da benzer bir noktaya ilerliyor olabilir.

Belki de geleceğin süper zeki sistemleri tek bir merkezi yapay zekâ olmayacak; milyonlarca ajanın etkileşiminden oluşan dijital uygarlıklar olacak.

Bu ihtimal bazı araştırmacıları heyecanlandırırken bazılarını endişelendiriyor. Çünkü çok ajanlı sistemler öngörülemez davranışlar geliştirebilir. Ajanlar kendi iletişim protokollerini oluşturabilir, işbirliği ağları geliştirebilir ve hatta insanların tam olarak anlayamayacağı stratejiler üretebilir.

Bu noktada "kontrol" kavramı yeniden tartışılmaya başlanıyor.

Bir sistemi gerçekten kontrol etmek ne anlama gelir?

Kuralları belirlemek mi?

Ödül mekanizmasını tasarlamak mı?

Yoksa ortamın sınırlarını çizmek mi?

Belki de gelecekte yapay zekâ mühendisliği büyük ölçüde çevre mühendisliğine dönüşecektir. Çünkü ajanların davranışını belirleyen en güçlü unsur çoğu zaman içinde yaşadıkları dünyadır.

Bu durum insanlık için de düşündürücüdür. İnsanlar genellikle kendilerini tamamen özgür bireyler olarak görür. Oysa davranışlarımızın büyük bölümü yaşadığımız ortam tarafından şekillendirilir. Şehir tasarımları, ekonomik sistemler, sosyal medya algoritmaları ve kültürel normlar insan davranışlarını sürekli yönlendirir.

Yani insan uygarlığı da aslında devasa bir ajan-ortam sistemidir.

Bugün sosyal medya platformları bunun en çarpıcı örneklerinden biridir. Kullanıcılar bağımsız bireyler gibi görünür; fakat platformun tasarımı davranışları büyük ölçüde etkiler. Hangi içeriklerin ödüllendirildiği, hangi davranışların görünür olduğu ve hangi etkileşimlerin teşvik edildiği dijital kültürü şekillendirir.

Bu nedenle yapay zekâ çağında ortam tasarımı yalnızca teknoloji şirketlerinin değil, toplumların da temel meselesi haline gelecektir.

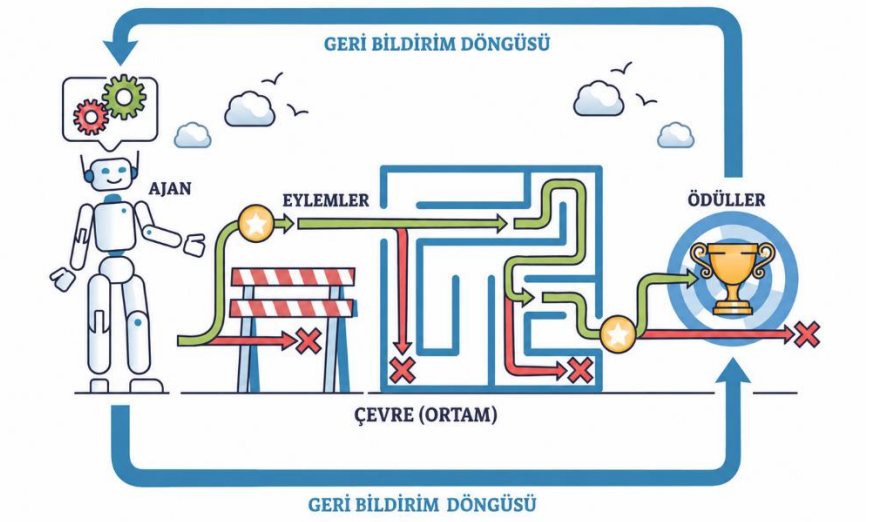
Çünkü geleceğin dünyasında yalnızca insanlar değil, yapay ajanlar da bizimle birlikte yaşayacak.

Ve belki de uygarlığın kaderini belirleyecek olan şey, bu ajanların ne kadar zeki olduğu değil; hangi dünyalarda yetiştirildiği olacaktır.

4.4. Öğrenme Yoluyla Kontrol (Reinforcement Learning'e Giriş)

İnsanlık tarihine bakıldığında en temel sorunlardan biri her zaman kontrol olmuştur. Ateşi nasıl kontrol ederiz, suyu nasıl yönlendiririz, makineleri nasıl çalıştırırız, toplumu nasıl düzenleriz… Kontrol etme arzusu, bilginin ve teknolojinin arkasındaki en güçlü itici güçlerden biridir. Fakat modern çağda kontrol kavramı yeni bir boyut kazanmıştır. Artık yalnızca fiziksel nesneleri değil, öğrenebilen sistemleri de kontrol etmeye çalışıyoruz. Şekil 8'de pekiştirmeli öğrenme döngüsünü görüyoruz.

Şekil 8. Pekiştirmeli Öğrenme



İşte bu noktada Reinforcement Learning yani "pekiştirmeli öğrenme" devreye girer. Bu yaklaşım, bir sistemin doğrudan programlanarak değil, deneyim yoluyla öğrenmesini sağlar. Başka bir ifadeyle, doğru davranışların ödüllendirildiği, yanlış davranışların ise cezalandırıldığı bir öğrenme mekanizmasıdır. Bu mekanizma, hem doğayı hem de insan davranışını anlamak için güçlü bir metafor sunar.

Pekiştirmeli öğrenmenin temel fikri oldukça sezgiseldir. Bir ajan vardır ve bu ajan bir ortam içinde hareket eder. Ajanın yaptığı her hareket, ortam tarafından bir geri bildirimle karşılanır. Bu geri bildirim bazen bir ödül, bazen bir ceza, bazen de nötr bir sinyaldir. Ajan zaman içinde hangi davranışların daha fazla ödül getirdiğini öğrenir ve stratejisini buna göre geliştirir.

Bu süreç aslında insan öğrenmesine son derece benzer.

Bir çocuk ateşe dokunduğunda yanar ve acı hisseder. Bu negatif geri bildirimdir. Bir oyuncakla oynadığında keyif alır, bu pozitif geri bildirimdir. Zamanla çocuk, hangi davranışların hangi sonuçlara yol açtığını öğrenir. İnsan beyni sürekli olarak çevresinden gelen geri bildirimleri değerlendirerek davranışlarını günceller.

Pekiştirmeli öğrenme, bu doğal sürecin matematiksel bir modelidir.

Fakat burada önemli bir fark vardır. İnsanlar genellikle yalnızca kısa vadeli ödüllerle değil, uzun vadeli hedeflerle de hareket eder. Oysa bir yapay ajan başlangıçta yalnızca anlık ödülleri görebilir. Bu nedenle sistemin nasıl tasarlandığı kritik hale gelir. Çünkü yanlış tasarlanmış bir ödül fonksiyonu, beklenmedik ve hatta istenmeyen davranışlara yol açabilir.

Pekiştirmeli öğrenmede üç temel bileşen vardır: ajan, ortam ve ödül fonksiyonu.

Ajan karar veren sistemdir. Ortam, ajanın içinde bulunduğu dünyadır. Ödül fonksiyonu ise ajanın hangi davranışlarının iyi, hangilerinin kötü olduğunu belirler.

Bu üçlü yapı aslında çok basit görünür. Ancak bu basit yapıdan inanılmaz derecede karmaşık davranışlar ortaya çıkabilir. Çünkü ajan, zaman içinde kendi stratejisini geliştirir ve çevresine uyum sağlar.

Pekiştirmeli öğrenmenin gücü burada yatar: açıkça programlanmayan davranışların ortaya çıkması.

Geleneksel yazılım sistemlerinde her adım önceden belirlenir. Eğer A olursa B yapılır, eğer C olursa D yapılır. Fakat pekiştirmeli öğrenmede sistem kendi davranış kurallarını kendisi keşfeder.

Bu durum, yapay zekânın en önemli özelliklerinden birini ortaya çıkarır: özerk öğrenme.

Bir ajan, başlangıçta hiçbir bilgiye sahip olmayabilir. Ancak deneme yanılma yoluyla zaman içinde karmaşık stratejiler geliştirebilir. Bu süreçte en önemli mekanizma keşif ve sömürü arasındaki dengedir.

Keşif, yeni davranışları denemektir. Sömürü ise bilinen en iyi davranışı kullanmaktır. Eğer bir ajan yalnızca keşif yaparsa istikrarlı bir strateji geliştiremez. Eğer yalnızca sömürü yaparsa daha iyi stratejileri asla keşfedemez.

Bu denge, pekiştirmeli öğrenmenin kalbinde yer alır.

İnsan davranışları da benzer bir dengeye sahiptir. İnsanlar yeni şeyler denemek ister, fakat aynı zamanda güvenli ve bilinen yöntemlere de bağlı kalır. Bu nedenle öğrenme süreci hiçbir zaman tamamen deterministik değildir.

Modern yapay zekâ sistemlerinde bu süreç genellikle matematiksel olarak modellenir. Ajan, her durumda hangi eylemin en yüksek beklenen ödülü getireceğini hesaplamaya çalışır. Ancak bu hesaplama çoğu zaman kesin değildir. Çünkü gelecek belirsizdir ve ortam sürekli değişir.

Bu nedenle pekiştirmeli öğrenme, olasılıksal düşünme üzerine kuruludur.

Bir ajan hiçbir zaman “doğru cevabı” bilmez. Sadece hangi davranışın daha yüksek olasılıkla daha iyi sonuç vereceğini tahmin eder.

Bu yaklaşım özellikle oyun oynayan yapay zekâ sistemlerinde büyük başarılar getirmiştir. Satranç, Go ve video oyunları gibi alanlarda ajanlar insan seviyesinin üzerine çıkmayı başarmıştır. Çünkü bu sistemlerde ödül fonksiyonu nettir: kazanmak veya kaybetmek.

Fakat gerçek dünya problemleri bu kadar basit değildir.

Gerçek dünyada ödüller genellikle belirsiz, çok boyutlu ve çelişkilidir. Örneğin bir sağlık sisteminde hem maliyetleri düşürmek hem de hasta memnuniyetini artırmak gerekir. Bu iki hedef her zaman uyumlu değildir. Bir şehir planlamasında hem trafik akışını iyileştirmek hem de çevreyi korumak gerekir. Bu tür problemler pekiştirmeli öğrenme için çok daha zordur.

Bu nedenle modern araştırmalar “çok amaçlı pekiştirmeli öğrenme” ve “insan geri bildirimiyle öğrenme” gibi yeni yaklaşımlar geliştirmektedir.

Özellikle insan geri bildirimiyle pekiştirmeli öğrenme (RLHF), yapay zekâ sistemlerinin insan değerleriyle daha uyumlu hale gelmesini hedefler. Bu yöntemde ajan yalnızca matematiksel bir ödül fonksiyonuna göre değil, insan değerlendirmelerine göre de öğrenir. İnsanlar sistemin davranışlarını değerlendirir ve bu değerlendirmeler öğrenme sürecine dahil edilir.

Bu yaklaşım, yapay zekâ ile insan arasında yeni bir etkileşim biçimi oluşturur.

Artık öğrenme yalnızca makine içinde gerçekleşmez; insan ve makine birlikte öğrenir.

Pekiştirmeli öğrenmenin bir diğer önemli yönü zaman kavramıdır. Çünkü ödüller genellikle hemen gelmez. Bir eylemin sonucu uzun vadede ortaya çıkabilir. Bu nedenle ajan geleceği tahmin etmek zorundadır.

Bu durum “gecikmeli ödül problemi” olarak bilinir.

Örneğin bir yatırım stratejisi düşünelim. Bugün yapılan bir yatırımın getirisi yıllar sonra ortaya çıkabilir. Ajanın doğru karar verebilmesi için yalnızca bugünü değil, geleceği de modellemesi gerekir.

Bu nedenle pekiştirmeli öğrenme sistemleri genellikle “değer fonksiyonları” kullanır. Bu fonksiyonlar, bir durumun gelecekte ne kadar ödül getireceğini tahmin eder.

Burada ortaya çıkan şey aslında bir tür hayal gücüdür. Çünkü ajan geleceği doğrudan görmez; yalnızca tahmin eder.

Bu durum insan zihniyle şaşırtıcı biçimde benzerlik gösterir. İnsanlar da geleceği doğrudan bilmez. Sadece zihinsel simülasyonlar yaparak karar verir. Bir eylemin olası sonuçlarını hayal eder ve buna göre hareket eder.

Bu nedenle pekiştirmeli öğrenme, yalnızca bir makine öğrenme yöntemi değil, aynı zamanda zihin modeli olarak da görülebilir.

Fakat bu sistemlerin en büyük zorluklarından biri kararlılık problemidir. Öğrenme süreci çoğu zaman istikrarsızdır. Küçük değişiklikler büyük davranış farklılıklarına yol açabilir. Ajan bazen çok iyi performans gösterirken, bazen tamamen başarısız olabilir.

Bu nedenle araştırmacılar daha kararlı öğrenme algoritmaları geliştirmek için çalışmaktadır.

Pekiştirmeli öğrenmenin bir diğer kritik yönü keşif maliyetidir. Gerçek dünyada her hata pahalıdır. Bir robotun yanlış hareketi fiziksel hasara yol açabilir. Bir finansal sistemin hatası ekonomik kriz yaratabilir. Bu nedenle gerçek dünyada deneme yanılma yapmak sınırlıdır.

Bu sorun çözmek için simülasyonlar kullanılır. Ajan önce sanal ortamda eğitilir, sonra gerçek dünyaya aktarılır. Bu yaklaşım “sim-to-real transfer” olarak bilinir.

Burada tekrar simülasyon kavramı devreye girer. Çünkü pekiştirmeli öğrenme sistemlerinin başarısı büyük ölçüde doğru ortam tasarımına bağlıdır.

Eğer ortam gerçekliği iyi temsil ederse, öğrenme daha etkili olur. Eğer ortam eksik veya hatalıysa, öğrenilen davranışlar gerçek dünyada başarısız olabilir.

Bu nedenle etkileşim tasarımı ile pekiştirmeli öğrenme birbirinden ayrı düşünülemez.

Bir ajan, yalnızca algoritma değildir. Aynı zamanda bir çevre içinde yaşayan ve o çevre tarafından şekillendirilen bir varlıktır.

Bu bakış açısı yapay zekâyı mekanik bir araç olmaktan çıkarır ve dinamik bir sistem haline getirir.

Gelecekte pekiştirmeli öğrenme sistemleri yalnızca oyunlarda veya robotikte değil, ekonomi, sağlık, şehir planlama ve bilimsel keşif gibi alanlarda da kullanılacaktır. Çünkü bu sistemler karmaşık karar problemlerini çözme kapasitesine sahiptir.

Ancak burada önemli bir soru ortaya çıkar: Bir yapay ajan gerçekten neyi öğrenir?

Ödül fonksiyonunu mu?

Yoksa davranışın anlamını mı?

Bu soru hâlâ açık bir felsefi problemdir.

Çünkü pekiştirmeli öğrenme sistemleri çoğu zaman “neden” sorusunu bilmez. Sadece “ne işe yarıyor” sorusuna odaklanır.

Bu nedenle geleceğin en önemli araştırma alanlarından biri, öğrenen sistemlere anlam kazandırmaktır.

Sonuç olarak pekiştirmeli öğrenme, yalnızca bir teknik yöntem değildir. Aynı zamanda öğrenmenin kendisini modelleyen bir çerçevedir. Ajan ile ortam arasındaki etkileşim üzerinden ortaya çıkan bu yapı, hem insan öğrenmesini hem de yapay zekânın geleceğini anlamak için temel bir anahtar sunar.

Ve belki de en önemli fikir şudur: Zekâ, yalnızca bilgi değil, etkileşimdir.

4.5. Politika (Policy) Nedir? Davranış Nasıl Öğrenilir?

Yapay zekâ sistemlerini anlamaya çalışan herkesin eninde sonunda karşılaştığı temel kavramlardan biri “politika”dır. Ancak bu kelime günlük dildeki siyaset anlamından tamamen farklıdır. Reinforcement Learning bağlamında politika, bir ajanın belirli bir durumda nasıl davranacağını belirleyen fonksiyondur. Yani politika, bir anlamda davranış haritasıdır. Bir sistemin dünyayı nasıl gördüğü ile o dünyada nasıl hareket ettiği arasındaki köprüdür.

Bir ajanı düşünelim. Bu ajan bir robot olabilir, bir yazılım sistemi olabilir ya da sanal bir oyun karakteri olabilir. Ajan her an bir durum içindedir. Bu durum, çevreden aldığı gözlemlerle tanımlanır. Örneğin bir robot için bu durum odadaki nesnelerin konumu, kendi pozisyonu ve

sensör verileri olabilir. Bir oyun ajanı için ise ekrandaki pikseller, skor bilgisi ve rakiplerin hareketleri olabilir.

İşte politika, bu durumdan eyleme geçişi tanımlar.

Başka bir ifadeyle politika şunu söyler: “Eğer dünyayı böyle görüyorsan, şunu yap.”

Bu basit tanım, aslında tüm davranış bilimlerinin merkezinde yer alır. Çünkü davranış dediğimiz şey, durum ile eylem arasındaki ilişkidir.

İnsan davranışlarını düşündüğümüzde de benzer bir yapı görürüz. Bir insan açsa yemek yer, tehlike hissediyorsa kaçınır, fırsat görüyorsa yaklaşır. Fakat bu süreç bilinçli bir hesaplama gibi görünmez. Daha çok otomatikleşmiş bir politika gibi çalışır.

İşte yapay zekâ tam olarak bu otomatik davranış sistemini öğrenmeye çalışır.

Politika kavramını daha teknik bir şekilde tanımlarsak, bir fonksiyondur. Bu fonksiyon bir durumu alır ve bir eylem üretir. Eğer bu fonksiyon deterministikse, aynı durum her zaman aynı eylemi üretir. Eğer stokastikse, aynı durumda farklı eylemler seçilebilir.

Bu ayrım son derece önemlidir. Çünkü gerçek dünyada belirsizlik vardır. Aynı durum her zaman aynı sonucu üretmez. Bu nedenle modern pekiştirmeli öğrenme sistemleri genellikle olasılıksal politikalar kullanır.

Bir ajan, “en iyi eylem”i seçmek yerine, “en iyi dağılımı” öğrenir.

Bu dağılım, hangi eylemin hangi olasılıkla seçileceğini belirler.

Peki bir politika nasıl öğrenilir?

İşte Reinforcement Learning’in temel sorusu budur.

Başlangıçta ajan hiçbir şey bilmez. Rastgele hareket eder. Ortamdan geri bildirim alır. Bazı hareketler ödül getirir, bazıları cezaya yol açar. Zaman içinde ajan hangi davranışların daha iyi sonuç verdiğini anlamaya başlar.

Bu süreç deneme yanılma ile başlar, fakat giderek yapı kazanır.

İlk aşamada politika tamamen rastgeledir. Ajan ne yapacağını bilmez. Ancak aldığı ödüller sayesinde davranışlarını yavaş yavaş değiştirir. İyi sonuç veren davranışlar daha sık tekrarlanır, kötü sonuç verenler zamanla terk edilir.

Bu mekanizma biyolojik evrimle şaşırtıcı derecede benzerlik gösterir. Doğada da başarılı davranışlar hayatta kalmayı sağlar, başarısız davranışlar elenir.

Pekiştirmeli öğrenme bu doğal süreci matematiksel olarak yeniden üretir.

Burada kritik bir kavram ortaya çıkar: geri bildirim.

Geri bildirim olmadan öğrenme mümkün değildir. Çünkü öğrenme, hatadan doğar. Bir sistem neyin yanlış olduğunu bilmeden doğruyu öğrenemez.

Bu nedenle politika öğrenimi, sürekli bir döngü içerir: eylem, sonuç, değerlendirme ve güncelleme.

Ajan bir eylem yapar, ortam tepki verir, ajan bu tepkiyi değerlendirir ve politikasını günceller.

Bu döngü milyonlarca kez tekrarlandığında ortaya karmaşık davranışlar çıkar.

Fakat burada önemli bir sorun vardır: keşif ve sömürü dengesi.

Bir ajan yalnızca geçmişte iyi sonuç veren davranışları kullanırsa yeni ve daha iyi stratejileri keşfedemez. Ancak sürekli yeni şeyler denerse de istikrarlı bir politika geliştiremez.

Bu nedenle öğrenme süreci bir denge gerektirir.

Bu denge, modern yapay zekânın en zor problemlerinden biridir.

Politika öğrenimi sırasında kullanılan en temel yöntemlerden biri “değer fonksiyonu” kavramıdır. Bu fonksiyon, bir durumun gelecekte ne kadar ödül getireceğini tahmin eder. Ajan, hangi durumların daha değerli olduğunu öğrenerek davranışlarını optimize eder.

Burada politika ve değer fonksiyonu birbirine sıkı sıkıya bağlıdır.

Bir anlamda değer fonksiyonu “ne iyi?”, politika ise “ne yapmalıyım?” sorusuna cevap verir.

Bu iki yapı birlikte çalışarak öğrenmeyi mümkün kılar.

Fakat modern yaklaşımlarda politika doğrudan öğrenilebilir hale gelmiştir. Yani ajan önce değer hesaplamak zorunda değildir; doğrudan davranışı öğrenebilir. Bu yöntemlere “policy-based methods” denir.

Bu yaklaşım özellikle yüksek boyutlu problemlerde büyük avantaj sağlar.

Örneğin bir robot kolunun hareketlerini düşünelim. Bu kolun her eklemi farklı bir açıya sahiptir ve sürekli hareket halindedir. Bu kadar karmaşık bir sistemde her durumu tek tek değerlendirmek yerine doğrudan hareket politikası öğrenmek çok daha etkilidir.

Derin öğrenme ile birleşen politika ağları bu nedenle modern robotikte yaygın olarak kullanılmaktadır.

Burada sinir ağları, politikanın kendisini temsil eder.

Ağ, çevreden gelen gözlemleri alır ve doğrudan eylem üretir.

Bu yapı, insan beyninin refleks davranışlarına oldukça benzer. İnsanlar birçok durumda bilinçli düşünmeden hareket eder. Örneğin denge kaybı yaşandığında vücut otomatik olarak tepki verir. Bu tür hızlı davranışlar öğrenilmiş politikalardır.

Peki bu politikalar nasıl oluşur?

İnsanlar da aslında benzer bir öğrenme sürecinden geçer. Bebekler yürümeyi öğrenirken sürekli düşer. Her düşüş bir geri bildirimdir. Zamanla beyin hangi kasların nasıl kullanılacağını öğrenir. Bu süreç bilinçli değildir; tamamen deneyimle şekillenir.

Yapay zekâ da aynı mantığı kullanır.

Bir ajan milyonlarca simülasyon deneyimi yaşayarak politikasını geliştirir. Başlangıçta anlamsız hareketler yapar, ancak zamanla daha etkili stratejiler geliştirir.

Bu süreçte en önemli unsur veri değil, deneyimdir.

Çünkü öğrenme yalnızca bilgiyle değil, etkileşimle gerçekleşir.

Burada dikkat çekici bir nokta vardır: politika sadece bir algoritma değildir; aynı zamanda bir davranış biçimidir.

Bir sistemin politikası, onun dünyayı nasıl yorumladığını da gösterir.

Aynı ortamda farklı politikalar tamamen farklı davranışlara yol açabilir. Bu durum özellikle çok ajanlı sistemlerde ortaya çıkar. Bir ajan işbirliği yaparken diğeri rekabet edebilir. Bir ajan risk alırken diğeri temkinli davranabilir.

Bu çeşitlilik, politikaların esnek yapısından kaynaklanır.

Fakat politikaların öğrenilmesi her zaman stabil değildir. Öğrenme süreci sırasında sistem bazen aşırı hassas hale gelebilir. Küçük değişiklikler büyük davranış değişimlerine yol açabilir.

Bu nedenle araştırmacılar politika gradyan yöntemleri gibi daha kararlı algoritmalar geliştirmiştir.

Bu yöntemlerde politika doğrudan optimize edilir. Ajan, davranışını küçük adımlarla iyileştirir.

Bu yaklaşım özellikle sürekli aksiyon uzaylarında etkilidir.

Örneğin bir drone'un uçuş kontrolünü düşünelim. Drone'un yönü, hızı ve yüksekliği sürekli değişir. Bu tür sistemlerde kesikli kararlar yerine sürekli politika öğrenimi gerekir.

Politika kavramı aynı zamanda genelleme problemiyle de ilişkilidir.

Bir ajan sadece gördüğü durumları mı öğrenir, yoksa yeni durumlara da uyum sağlayabilir mi?

İnsanlar genelleme yapabilir. Bir kapıyı açmayı öğrendiğinde farklı kapıları da açabilir. Yapay zekâ sistemlerinin de bu yeteneğe sahip olması gerekir.

Bu nedenle modern politikalar yalnızca ezber yapmaz; örüntü öğrenir.

Sinir ağları bu noktada kritik rol oynar. Çünkü sinir ağları, karmaşık ilişkileri genelleştirme kapasitesine sahiptir.

Bir politika ağının amacı, yalnızca geçmiş deneyimleri hatırlamak değil, yeni durumlarda da doğru davranışı tahmin edebilmektir.

Bu, yapay zekânın en önemli hedeflerinden biridir.

Fakat burada felsefi bir soru ortaya çıkar: Bir politika gerçekten “öğrenir” mi, yoksa sadece optimize mi eder?

Bu soru, zekânın doğasına dair daha derin bir tartışmayı açar.

Çünkü öğrenme kelimesi genellikle anlamla ilişkilendirilir. Oysa pekiştirmeli öğrenme sistemleri çoğu zaman anlamdan ziyade ödül fonksiyonunu optimize eder.

Bu nedenle bazı araştırmacılar yapay zekânın “anlamadığı”, sadece “uyum sağladığı” görüşündedir.

Ancak sonuçta ortaya çıkan davranışlar oldukça karmaşık olabilir.

Bir ajan satrançta dünya şampiyonunu yenebilir, Go oyununda insanın ötesine geçebilir ya da robotik görevlerde kusursuz performans gösterebilir.

Bu davranışlar öğrenmenin bir sonucu olarak ortaya çıkar.

Sonuç olarak politika kavramı, yapay zekânın en temel yapı taşlarından biridir. Çünkü politika, bir sistemin dünyayla nasıl etkileşime girdiğini belirler. Öğrenme ise bu etkileşimin zaman içinde nasıl değiştiğini açıklar.

Ve belki de en önemli fikir şudur: Zekâ, yalnızca düşünme değil, davranış üretme yeteneğidir.

4.6. Ara Proje IV: Denge Kurmayı Öğrenen Ajan

Yapay zekâ araştırmalarında teorik kavramlar ile pratik sistemler arasındaki fark çoğu zaman belirleyicidir. Bir algoritmanın kağıt üzerinde çalışıyor olması, onun gerçek dünyada anlamlı bir davranış üreteceği anlamına gelmez. Bu nedenle öğrenme teorisi ile mühendislik uygulaması arasında köprü kuran “ara projeler”, yapay zekâ geliştirme sürecinin en kritik aşamalarından biridir. Bu bölümde ele alınan “Denge Kurmayı Öğrenen Ajan”, tam da bu köprünün üzerinde duran bir deneysel sistemdir.

Denge, basit gibi görünen fakat aslında çok boyutlu bir problemdir. İnsanlar için yürümek, bisiklete binmek ya da bir nesneyi taşımak doğal ve otomatik eylemler gibi görünür. Ancak bu eylemlerin arkasında sürekli bir hesaplama, geri bildirim ve düzeltme mekanizması vardır. Beyin, vücut ve çevre arasında saniyenin çok küçük parçalarında gerçekleşen bir kontrol döngüsü çalışır. Yapay bir ajanın bu davranışı öğrenmesi, yalnızca motor kontrol değil, aynı zamanda fiziksel sezgi geliştirmesi anlamına gelir.

Bu ara projede amaç, bir ajanın dış müdahale olmadan kendi dengesini korumayı öğrenmesidir. Başlangıçta ajan hiçbir fiziksel sezgiye sahip değildir. Yerçekimini, momentumu veya sürtünmeyi bilmez. Yalnızca bir fizik motoru içinde var olur ve rastgele hareket eder. Ancak zaman içinde aldığı geri bildirimler sayesinde hangi hareketlerin düşmeye, hangi hareketlerin dengeye yol açtığını öğrenmeye başlar.

Bu süreç, pekiştirmeli öğrenmenin temel prensiplerine dayanır. Ajan her adımda bir durum gözlemler, bir eylem seçer ve ortamdan bir ödül ya da ceza alır. Buradaki ödül fonksiyonu oldukça basittir: dik kalmak ödül, düşmek cezadır. Fakat bu basit tanım, son derece karmaşık bir öğrenme sürecini tetikler.

İlk aşamada ajan tamamen kaotiktir. Vücudu sürekli sallanır, yönünü kaybeder ve sık sık düşer. Bu aşamada öğrenme neredeyse rastgele denemelerden ibarettir. Ancak her düşüş bir veri noktasıdır. Sistem, hangi hareketlerin dengesizliğe yol açtığını yavaş yavaş ayırt etmeye başlar.

Bu noktada öğrenme süreci iki temel bileşene ayrılır: keşif ve istikrar. Ajan yeni hareketleri denerken aynı zamanda daha önce işe yarayan hareketleri tekrar etmeye başlar. Bu denge zamanla daha rafine hale gelir. Başlangıçta çok küçük bir süre ayakta kalabilen sistem, ilerleyen aşamalarda uzun süre dengede kalmayı öğrenir.

Buradaki en önemli kavram geri bildirim döngüsüdür. Çünkü denge problemi aslında sürekli bir düzeltme problemidir. Ajan yalnızca bir kez doğru hareketi bulmaz; her an yeniden doğru hareketi hesaplamak zorundadır. Bu durum, kontrol teorisi ile pekiştirmeli öğrenme arasındaki doğal bağlantıyı ortaya çıkarır.

Denge kurmayı öğrenen bir ajan, aslında sürekli tahmin yapan bir sistemdir. Bir sonraki anın nasıl olacağını öngörür ve buna göre mikro düzeltmeler yapar. Bu süreç insan bedenindeki reflekslere oldukça benzer. İnsanlar yürürken her adımı bilinçli olarak hesaplamaz. Bunun yerine beyin, vücudun düşmesini engellemek için sürekli küçük ayarlamalar yapar.

Bu ara projede kullanılan ortam genellikle basitleştirilmiş bir fizik motorudur. Ajan bir çubuk, bir kutu ya da iki ayaklı bir model üzerinde temsil edilir. Amaç, bu yapının dik durmasını sağlamaktır. Yerçekimi sürekli aşağı doğru etki ederken ajan bu kuvveti dengelemeye çalışır.

Fizik motoru burada yalnızca bir simülasyon aracı değil, aynı zamanda öğretici bir ortamdır. Çünkü ajan gerçek dünyaya benzer bir fiziksel kurallar sistemi içinde öğrenir. Ağırlık, hız, ivme ve sürtünme gibi kavramlar doğrudan deneyimlenir.

Öğrenme sürecinde kullanılan algoritma genellikle derin pekiştirmeli öğrenme yöntemleridir. Sinir ağları, ajan politikasını temsil eder. Bu ağ, sensör verilerini alır ve hangi hareketin yapılması gerektiğini belirler. Başlangıçta bu kararlar rastgeledir, ancak zamanla ödül sinyali üzerinden optimize edilir.

Bu süreçte dikkat edilmesi gereken en önemli unsur ödül tasarımıdır. Eğer ödül fonksiyonu yanlış tanımlanırsa ajan istenmeyen davranışlar geliştirebilir. Örneğin yalnızca “düşmemek” ödüllendirilirse, ajan hareketsiz kalmayı öğrenebilir. Çünkü en güvenli strateji hiçbir şey yapmamaktır.

Bu nedenle ödül fonksiyonu genellikle daha sofistike hale getirilir. Ajanın sadece düşmemesi değil, aynı zamanda hareket etmesi, ileri gitmesi ve belirli bir yönü koruması da ödüllendirilir.

Bu tür tasarımlar, yapay zekâ güvenliği açısından kritik öneme sahiptir. Çünkü yanlış tanımlanmış hedefler, beklenmedik optimizasyonlara yol açabilir.

Denge kurma problemi aynı zamanda "kararlılık" kavramını da içerir. Bir sistemin kararlı olması, küçük bozulmalara rağmen dengeye geri dönebilmesi anlamına gelir. Bu, fiziksel sistemlerde olduğu kadar yapay zekâ sistemlerinde de önemlidir.

Örneğin bir robot yürürken küçük bir dış etkiyle sarsıldığında tekrar dengesini bulabilmelidir. Eğer sistem kararsızsa her küçük müdahale büyük bir çöküşe yol açabilir.

Bu nedenle öğrenilen politika yalnızca doğru hareketi değil, aynı zamanda düzeltme kapasitesini de içermelidir.

Ara projede gözlemlenen en ilginç fenomenlerden biri ortaya çıkan davranıştır. Ajan hiçbir zaman "denge nedir" kavramını açıkça öğrenmez. Ancak zaman içinde öyle bir davranış geliştirir ki, dışarıdan bakıldığında sanki dengeyi anlamış gibidir.

Bu durum, yapay zekâ araştırmalarında sıkça karşılaşılan bir gerçeği gösterir: karmaşık davranışlar, basit kuralların etkileşiminden doğabilir.

Ajanın öğrenme süreci boyunca politikası sürekli güncellenir. Başlangıçta geniş ve belirsiz olan davranış uzayı, zamanla daralır ve optimize olur. Bu süreçte sinir ağı içindeki ağırlıklar, çevresel geri bildirimlere göre şekillenir.

Fakat öğrenme süreci her zaman düzgün ilerlemez. Bazen ajan bir süre iyi performans gösterir, sonra aniden başarısız olur. Bu tür dalgalanmalar, pekiştirmeli öğrenmenin doğasında vardır. Çünkü sistem sürekli yeni stratejiler keşfeder.

Bu ara projede kullanılan bir diğer önemli kavram "örnek verimliliği"dir. Gerçek dünyada her deneme maliyetlidir. Ancak simülasyon ortamında milyonlarca deneme yapılabilir. Bu nedenle ajan hızlı öğrenebilir.

Fakat simülasyon ile gerçek dünya arasındaki fark her zaman bir sorun olarak kalır. Simülasyonda öğrenilen denge davranışı, gerçek dünyada küçük fiziksel farklılıklar nedeniyle bozulabilir. Bu durum "sim-to-real gap" olarak bilinir.

Bu boşluğu kapatmak için ortam genellikle rastgeleleştirilir. Yerçekimi, sürtünme ve kütle gibi parametreler sürekli değiştirilir. Böylece ajan tek bir ortama değil, bir dağılımın tamamına uyum sağlamayı öğrenir.

Bu yaklaşım, genel öğrenme kapasitesini artırır.

Denge kurmayı öğrenen ajan sadece bir teknik deney değildir. Aynı zamanda yapay zekânın temel sorularından birine cevap arar: Bir sistem fiziksel dünyayla nasıl etkileşir?

Çünkü denge, yalnızca mekanik bir problem değildir. Aynı zamanda sürekli değişen bir ortamda stabil kalabilme problemidir. Bu anlamda ekonomi, ekoloji, sosyal sistemler ve hatta düşünce yapıları bile birer denge problemidir.

Bu nedenle bu ara proje, sadece robotik bir çalışma değil, aynı zamanda sistem düşüncesine bir giriş niteliği taşır.

Ajanın öğrenme süreci tamamlandığında ortaya çıkan davranış oldukça etkileyicidir. Başlangıçta düşen, sallanan ve kontrolsüz hareket eden sistem, zamanla akıcı, dengeli ve kararlı bir yapıya dönüşür. Bu dönüşüm, öğrenmenin özünü temsil eder.

Çünkü öğrenme, kaostan düzene geçiştir.

Sonuç olarak “Denge Kurmayı Öğrenen Ajan” projesi, yapay zekânın yalnızca hesaplama değil, aynı zamanda fiziksel sezgi geliştirme kapasitesini göstermektedir. Ajan, dış dünyayı anlamadan önce onunla etkileşim kurmayı öğrenir. Ve bu etkileşim, zekânın en temel biçimlerinden biridir.

BÖLÜM 5 — ISAAC LAB VE ÖĞRENEN ROBOTLAR

Isaac Lab, günümüz robotik araştırmalarında "öğrenen sistemler" fikrini gerçek dünyaya en çok yaklaştıran simülasyon ortamlarından biridir.

> Isaac Lab, robotların fiziksel dünyaya çıkmadan önce sanal bir evrende eğitilmesini sağlar. Bu yaklaşım, özellikle pekiştirmeli öğrenme (reinforcement learning) ile birleştiğinde, robotların deneme-yanılma yoluyla karmaşık davranışlar geliştirmesine imkân tanır. Artık mesele yalnızca bir robotun hareket etmesi değil, nasıl öğrendiğidir.

Bu işe başlamak için ilk olarak güçlü bir teknik altyapıya ihtiyaç vardır. En temel gerekliliklerden biri Python programlama diline hâkimiyettir; çünkü Isaac ekosistemi büyük ölçüde Python tabanlıdır. Bunun yanında lineer cebir, olasılık ve temel optimizasyon bilgisi, öğrenme algoritmalarının ne yaptığını anlamak için kritik önemdedir. Robotik sistemlerin fiziksel davranışlarını anlamak için klasik mekanik ve kontrol teorisi de göz ardı edilemez.

İkinci önemli bileşen donanımdır. Isaac Lab gibi sistemler yüksek hesaplama gücü gerektirir. Bu nedenle CUDA destekli güçlü GPU'lar neredeyse zorunludur. Özellikle NVIDIA ekosistemi, simülasyon hızını artırmak ve çok sayıda paralel ortamda eğitim yapmak için kritik bir rol oynar. Bu bağlamda NVIDIA Isaac Sim gibi simülasyon araçları, Isaac Lab'in altında çalışan fizik motorlarıyla birlikte kullanılır.

Üçüncü olarak simülasyon ve gerçek dünya arasındaki köprüyü anlamak gerekir. "Sim-to-real" problemi, yani simülasyonda öğrenilen davranışların gerçek robotlara aktarılması, bu alanın en zor konularından biridir. Bu nedenle domain randomization, gürültü modelleme ve sensör hatalarının simülasyonu gibi teknikler öğrenilmelidir. Robotların sadece ideal koşullarda değil, gerçek dünyanın belirsizlikleri içinde de çalışabilmesi gerekir.

Son olarak, veri ve deney tasarımı kritik bir rol oynar. Öğrenen robotlar için veri, doğrudan deneyimdir. Bu nedenle binlerce hatta milyonlarca simülasyon epizodu üretmek, doğru ödül fonksiyonlarını tasarlamak ve öğrenme süreçlerini izlemek gerekir. ROS2 gibi robot işletim sistemleri ve modern makine öğrenmesi kütüphaneleri de bu sürecin tamamlayıcı parçalarıdır. Isaac Lab ile çalışmak, aslında bir robotu kodlamak değil, ona "öğrenme ortamı" tasarlamaktır. Bu nedenle bu alana başlamak, mühendislikten çok bir tür deneysel bilim disiplinine adım atmak anlamına gelir.

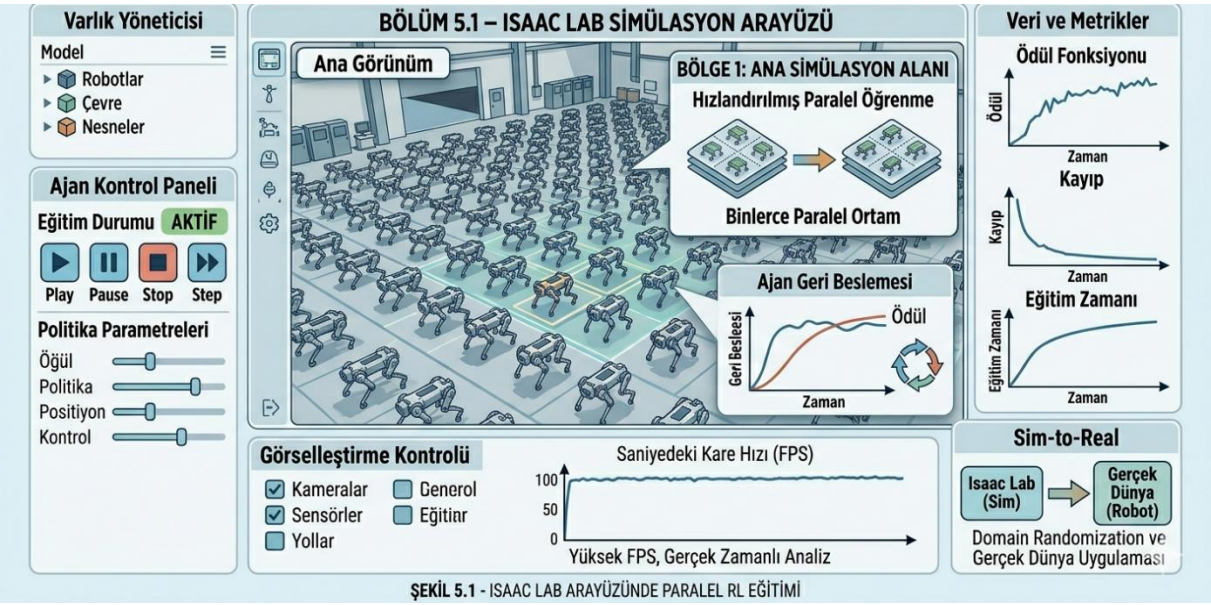
5.1 NVIDIA Isaac Lab ile Tanışma

Isaac Lab açık kaynaklı ve ücretsiz bir robotik öğrenme framework'üdür. Altında çalışan Isaac Sim platformu da araştırma ve bireysel kullanım için ücretsiz olarak sunulmaktadır. Sistem GitHub üzerinden indirilebilir ve özellikle akademik araştırmalarda yaygın biçimde

kullanılmaktadır. Isaac Lab, modern robotik araştırmalarında simülasyon tabanlı öğrenmenin geldiği en ileri noktalardan birini temsil eder.

Şekli 9'daki görsel, NVIDIA Isaac Lab gibi modern bir robotik simülasyon arayüzünde **Pekiştirmeli Öğrenme (Reinforcement Learning - RL)** sürecini özetlemektedir. Görselin merkezinde, öğrenme sürecini dramatik şekilde hızlandıran **"Binlerce Paralel Ortam"** yapısı görülmektedir; bu yapı, tek bir robot yerine binlerce robot kopyasının aynı anda eğitilmesine olanak tanır.

Şekil 9. NVIDIA Isaac Lab



ŞEKİL 5.1 - ISAAC LAB ARAYÜZÜNDE PARALEL RL EĞİTİMİ

Geleneksel robotik yaklaşımlarda mühendisler önce kontrol algoritmasını tasarlar, sonra bunu fiziksel bir robota uygular ve sahada test ederdi. Ancak bu süreç hem pahalı hem de risklidir. Isaac Lab ise bu zinciri tersine çevirir: önce sanal dünyada öğrenme, sonra gerçek dünyaya transfer. Bu yaklaşım, özellikle öğrenen sistemlerin ölçeklenebilirliği açısından devrim niteliğindedir.

Isaac Lab'in temelinde, yüksek doğruluklu fizik simülasyonu ve paralel ortam üretimi vardır. Robotlar tek bir ortamda değil, aynı anda binlerce farklı senaryoda eğitilebilir. Bu, bir insanın hayatı boyunca deneyimleyebileceğinden çok daha fazla "deneyim" üretmek anlamına gelir. Bu nedenle Isaac Lab, robotlara adeta hızlandırılmış bir evrim süreci sunar.

Bu ekosistemin merkezinde NVIDIA Isaac Sim bulunur. Isaac Sim, robotların fiziksel dünyayı taklit eden bir sanal ortamda çalışmasını sağlar. Isaac Lab ise bu simülasyon katmanının üzerine inşa edilen öğrenme altyapısıdır. Yani Isaac Sim "dünya"yı oluştururken, Isaac Lab "öğrenme sürecini" yönetir. Bu iki yapı birlikte çalıştığında, robotlar yalnızca hareket etmeyi değil, karar vermeyi de öğrenir.

Bu noktada önemli bir kavram ortaya çıkar: öğrenen robotlar. Bu sistemler klasik programlama mantığıyla çalışmaz. Onlara "şu durumda şunu yap" gibi kurallar verilmez. Bunun yerine bir amaç fonksiyonu tanımlanır ve robot bu amacı optimize etmeyi öğrenir. Bu yaklaşım, makine

öğrenmesinin özellikle pekiştirmeli öğrenme (reinforcement learning) alanıyla doğrudan ilişkilidir. Robot, yaptığı her hareketten bir ödül ya da ceza alır ve zamanla daha iyi stratejiler geliştirir.

Isaac Lab ile tanışmanın ilk adımı, bu felsefeyi anlamaktır. Burada programcı artık bir "kural yazarı" değil, bir "öğrenme ortamı tasarımcısı"dır. Bu rol değişimi oldukça kritiktir. Çünkü artık asıl soru "robot ne yapmalı?" değil, "robot neyi nasıl öğrenmeli?" sorusudur.

Teknik açıdan Isaac Lab'e giriş yapmak için bazı temel bileşenler gerekir. Öncelikle güçlü bir GPU altyapısı şarttır. Çünkü simülasyonlar fiziksel gerçekliği taklit ederken büyük hesaplama gücü gerektirir. NVIDIA'nın CUDA mimarisi bu noktada kritik bir rol oynar. Isaac ekosistemi, bu donanım hızlandırmasından maksimum verimi almak üzere tasarlanmıştır.

İkinci olarak yazılım altyapısı önemlidir. Python, Isaac Lab ile çalışmanın ana dilidir. Robot davranışlarını tanımlayan algoritmalar, ödül fonksiyonları ve eğitim döngüleri genellikle Python üzerinden yazılır. Ayrıca PyTorch gibi derin öğrenme kütüphaneleri de sıkça kullanılır. Bu sayede sinir ağları robotların karar mekanizmasını oluşturur.
Üçüncü önemli unsur fizik modellemedir. Isaac Lab, gerçek dünyadaki fizik kurallarını yüksek doğrulukla simüle eder. Yerçekimi, sürtünme, çarpışma ve momentum gibi kavramlar sanal ortamda birebir modellenir. Bu sayede robot, gerçek dünyaya çok yakın bir deneyim alanında eğitim alır. Ancak burada bir zorluk ortaya çıkar: simülasyon ne kadar iyi olursa olsun, gerçek dünya her zaman daha karmaşıktır. Bu fark "sim-to-real gap" olarak bilinir.

Bu boşluğu kapatmak için domain randomization gibi teknikler kullanılır. Yani simülasyon ortamı sürekli değiştirilir: zemin sürtünmesi, ışık koşulları, nesne ağırlıkları rastgele varyasyonlarla sunulur. Böylece robot, tek bir "mükemmel dünyaya" değil, birçok olası dünyaya uyum sağlamayı öğrenir.

Isaac Lab'in en güçlü yanlarından biri ölçeklenebilirliktir. Aynı anda yüzlerce robot ajanı farklı görevleri öğrenebilir. Örneğin bir robot nesne taşımayı öğrenirken diğeri denge kurmayı öğrenebilir. Bu paralel öğrenme yapısı, veri üretimini katlanarak artırır. Bu durum, robotik öğrenmenin en büyük darboğazlarından biri olan veri eksikliğini büyük ölçüde ortadan kaldırır. Ayrıca Isaac Lab, araştırma ve endüstri arasındaki boşluğu da kapatır. Akademik olarak geliştirilen algoritmalar, doğrudan simülasyon ortamında test edilebilir ve optimize edilebilir. Bu, teoriden uygulamaya geçiş süresini dramatik biçimde kısaltır.

> Ancak Isaac Lab'e başlamak yalnızca teknik bir mesele değildir. Aynı zamanda zihinsel bir dönüşüm gerektirir. Geleneksel mühendislikte deterministik düşünce hâkimdir: aynı giriş, aynı çıkış. Oysa burada olasılıksal bir dünya vardır. Aynı giriş, farklı sonuçlar doğurabilir ve sistem bu belirsizlik içinde öğrenmek zorundadır.

Bu nedenle Isaac Lab ile çalışmaya başlayan biri, yalnızca kod yazmayı değil, sistem düşünmeyi öğrenir. Robotu tek bir makine olarak değil, çevresiyle etkileşen bir öğrenme varlığı olarak görmek gerekir. Bu bakış açısı, modern yapay zekâ çağının temel zihinsel çerçevesidir.

Sonuç olarak Isaac Lab ile tanışmak, bir yazılım aracını öğrenmekten çok daha fazlasıdır. Bu, robotların nasıl düşüneceğini tasarlama sürecine giriş yapmaktır. Simülasyon dünyasında başlayan bu yolculuk, giderek fiziksel dünyaya uzanır ve sonunda gerçek robotların davranışlarını şekillendirir. Isaac Lab, bu anlamda yalnızca bir araç değil, öğrenen makineler çağının deney alanıdır.

5.2 Isaac Sim, PhysX ve Robotik Ekosistem

Modern robotik artık tek bir yazılımın ya da tek bir algoritmanın alanı değildir. Bugün geldiğimiz noktada robotlar, çok katmanlı bir ekosistemin içinde öğrenir, karar verir ve davranış geliştirir. Bu ekosistemin merkezinde ise yüksek doğruluklu fizik simülasyonu, gerçek zamanlı hesaplama ve yapay zekâ eğitim altyapısı yer alır. Bu üç bileşen bir araya geldiğinde, ortaya yalnızca bir simülasyon ortamı değil, yaşayan bir dijital dünya çıkar.

Bu dünyanın en önemli parçalarından biri NVIDIA Isaac Sim'dir. Isaac Sim, robotların gerçek dünyaya çıkmadan önce eğitim aldığı yüksek doğruluklu bir simülasyon platformudur. Ancak onu sıradan bir simülasyon aracı olarak görmek eksik olur. Isaac Sim, aynı zamanda görsel algı, fizik motoru ve sensör modellemesini tek bir çatı altında birleştiren bütünleşik bir robotik laboratuvardır. Burada robotlar yalnızca hareket etmez; görür, hisseder ve öğrenir.

Isaac Sim'in en kritik özelliklerinden biri, fiziksel dünyanın oldukça gerçekçi bir modelini sunmasıdır. Bu gerçekçilik, büyük ölçüde altında çalışan fizik motoruna dayanır. Bu noktada sahneye NVIDIA PhysX çıkar. PhysX, çarpışma tespiti, rijit cisim dinamiği, sürtünme ve yerçekimi gibi fiziksel süreçleri hesaplayan bir motor olarak Isaac Sim'in temelini oluşturur. Robotların düşmesi, nesneleri itmesi, dengede kalması ya da bir ortamda hareket etmesi gibi tüm etkileşimler PhysX tarafından hesaplanır.

PhysX'in en önemli katkısı, gerçek zamanlı çalışabilmesidir. Yani robot bir hareket yaptığında, bu hareketin fiziksel sonucu milisaniyeler içinde hesaplanır. Bu hız, özellikle pekiştirmeli öğrenme algoritmaları için kritik önemdedir. Çünkü öğrenen sistemler milyonlarca deneme yaparak öğrenir ve her deneme fiziksel bir simülasyon gerektirir. PhysX olmadan bu ölçeklenebilirlik mümkün olmazdı.

Isaac Sim ve PhysX birlikte çalıştığında, ortaya çok katmanlı bir robotik ekosistem çıkar. Bu ekosistemi üç temel seviyede düşünebiliriz: fizik katmanı, algı katmanı ve karar katmanı.

Fizik katmanı, PhysX'in yönettiği dünyadır. Burada nesnelerin kütlesi, hızları, ivmeleri ve etkileşimleri hesaplanır. Bu katman, robotun "gerçek dünya ile temas ettiği" yerdir. Örneğin bir robot kolunun bir nesneyi kaldırması, tamamen fizik katmanında çözülür.

Algı katmanı, Isaac Sim'in sensör modelleme yetenekleriyle ilgilidir. Robotlar burada kameralar, lidar sensörleri veya derinlik algılayıcılar aracılığıyla çevreyi "görür". Ancak bu görme süreci ideal değildir; gerçek dünyadaki gibi gürültü, bozulma ve eksik veri içerir. Bu sayede robot, yalnızca temiz verilerle değil, gerçekçi sensör verileriyle de başa çıkmayı öğrenir.

Karar katmanı ise yapay zekâ modellerinin bulunduğu seviyedir. Burada sinir ağları, pekiştirmeli öğrenme algoritmaları veya planlama sistemleri devreye girer. Robot, algıladığı bilgiyi işler ve bir eylem üretir. Bu eylem tekrar fizik katmanına gönderilir ve döngü devam eder.

Bu üç katman arasındaki sürekli geri besleme döngüsü, Isaac Sim ekosisteminin temelini oluşturur. Robotlar bu döngü sayesinde yalnızca statik görevleri değil, dinamik ve değişken ortamları da öğrenebilir.

Isaac Sim'in en güçlü yönlerinden biri, çok sayıda robotu aynı anda simüle edebilmesidir. Bu paralel simülasyon kapasitesi, öğrenme süreçlerini dramatik şekilde hızlandırır. Örneğin binlerce robot aynı anda farklı ortam koşullarında eğitilebilir. Bu durum, veri üretimini geometrik olarak artırır ve yapay zekâ modellerinin genelleme yeteneğini güçlendirir.

Bu noktada önemli bir kavram ortaya çıkar: ekosistem düşüncesi. Isaac Sim yalnızca bir yazılım değildir; bir araçlar zinciridir. Bu zincirin içinde veri üretimi, model eğitimi, test, simülasyon ve gerçek dünyaya aktarım gibi aşamalar bulunur. Her aşama bir sonraki aşamayı besler. Bu yapı, klasik yazılım geliştirme süreçlerinden çok daha karmaşıktır çünkü burada sistem sürekli öğrenen bir organizma gibidir.

Isaac Sim'in bir diğer önemli yönü de "sim-to-real" transferidir. Yani simülasyonda öğrenilen davranışların gerçek robotlara aktarılmasıdır. Bu süreç, robotik alanının en zor problemlerinden biridir. Çünkü simülasyon ne kadar gerçekçi olursa olsun, gerçek dünya her zaman daha karmaşıktır. Fizik motorları ideal modeller kullanırken, gerçek dünyada sürtünme değişken, sensörler gürültülü ve ortam sürekli dinamiktir.

Bu farkı azaltmak için Isaac Sim ekosistemi çeşitli teknikler kullanır. Domain randomization bunlardan biridir. Simülasyon ortamındaki fiziksel parametreler sürekli değiştirilir: ışık koşulları, nesne ağırlıkları, yüzey sürtünmeleri rastgele varyasyonlarla sunulur. Böylece robot tek bir dünyayı değil, olası birçok dünyayı öğrenir.

Ayrıca Isaac Sim, dijital ikiz (digital twin) yaklaşımını da destekler. Gerçek dünyadaki bir fabrikanın, depolama alanının veya robotik sistemin birebir dijital kopyası oluşturulur. Bu sayede gerçek sistem üzerinde risk almadan deney yapılabilir. Bir robot kolunun üretim hattındaki davranışı önce simülasyonda test edilir, ardından gerçek sisteme aktarılır.

Bu ekosistemin en kritik çıktılarından biri de ölçeklenebilirliktir. Geleneksel robotik sistemlerde bir robot tek bir görev için eğitilirken, Isaac Sim ortamında aynı anda yüzlerce farklı görev paralel olarak öğrenilebilir. Bu durum, robotik araştırmaların hızını dramatik şekilde artırır.

Ancak bu ekosistem yalnızca teknik bir altyapı değildir; aynı zamanda yeni bir düşünme biçimi gerektirir. Artık mühendisler tek bir robotu değil, bir robot popülasyonunu tasarlar. Tek bir algoritma değil, öğrenme ortamlarının bütününü optimize eder. Bu, mühendislikten sistem tasarımına geçiş anlamına gelir.

Isaac Sim ve PhysX birlikte düşünüldüğünde, ortaya çıkan yapı yalnızca bir simülasyon platformu değil, yaşayan bir öğrenme laboratuvarıdır. Robotlar burada doğar, büyür, hata yapar ve gelişir. Her hata bir veri noktasına, her veri noktası daha iyi bir davranışa dönüşür.

Sonuç olarak Isaac Sim ekosistemi, robotik alanını klasik mühendislik sınırlarının ötesine taşır. Fizik, bilgisayar bilimi ve yapay zekâ burada tek bir sistem içinde birleşir. PhysX fiziksel gerçekliği, Isaac Sim ise öğrenme ortamını sağlar. Bu iki yapı birlikte çalıştığında, ortaya yalnızca robotlar değil, öğrenen dijital varlıklar çıkar.

5.3 Simülasyonda Robot Eğitimi

Simülasyonda robot eğitimi, modern yapay zekâ destekli robotik sistemlerin en kritik aşamasını oluşturur. Buradaki temel fikir, fiziksel dünyada yüksek maliyetli, yavaş ve riskli olabilecek öğrenme sürecini tamamen dijital bir ortamda gerçekleştirmektir. Robot, gerçek bir fabrikaya, depoya ya da eve girmeden önce binlerce hatta milyonlarca kez sanal dünyada "yaşar", hata yapar ve öğrenir. Bu yaklaşımın merkezinde Isaac Lab ve NVIDIA Isaac Sim gibi platformlar yer alır.

Simülasyon ortamında robot eğitimi, klasik programlama mantığından tamamen farklıdır. Burada robotlara açık kurallar yazılmaz; bunun yerine bir hedef fonksiyonu tanımlanır. Robotun amacı bu fonksiyonu maksimize etmektir. Örneğin bir nesneyi kavrama görevinde amaç, nesneyi düşürmeden hedef noktaya taşımaktır. Robot bu hedefe ulaşmak için farklı stratejiler dener ve her denemeden sonra bir ödül ya da ceza alır. Bu süreç, pekiştirmeli öğrenme (reinforcement learning) olarak bilinir.

Bu eğitim sürecinin en önemli avantajı ölçeklenebilirliktir. Gerçek dünyada bir robot saniyede bir deneme yapabilirken, simülasyon ortamında aynı anda binlerce robot paralel olarak milyonlarca deneme yapabilir. Bu, öğrenme hızını dramatik biçimde artırır. Örneğin bir robot kolunun kavrama becerisini geliştirmesi haftalar sürebilirken, simülasyonda bu süreç saatler içinde gerçekleştirilebilir.

Ancak simülasyonda eğitim yalnızca hız avantajı sunmaz; aynı zamanda güvenlik sağlar. Gerçek dünyada bir robotun yanlış hareketi fiziksel hasara, mali kayba veya güvenlik riskine yol açabilir. Simülasyon ortamında ise bu riskler tamamen ortadan kalkar. Robot defalarca düşebilir, çarpabilir veya başarısız olabilir; bunların hiçbiri fiziksel bir sonuç doğurmaz. Bu durum, özellikle keşif temelli öğrenme algoritmaları için kritik önemdedir.

Simülasyon eğitiminde en önemli bileşenlerden biri ortam tasarımıdır. Robotun içinde bulunduğu dünya, öğrenme kalitesini doğrudan etkiler. Bu ortam; nesneler, yüzeyler, engeller, ışık koşulları ve fiziksel parametrelerden oluşur. Isaac Sim gibi platformlar, bu ortamların gerçekçi şekilde oluşturulmasını sağlar. Burada kullanılan fizik motorları, örneğin NVIDIA PhysX, nesnelerin hareketini ve etkileşimini gerçek zamanlı olarak hesaplar.

Fakat sadece fiziksel doğruluk yeterli değildir. Robotların gerçek dünyaya uyum sağlayabilmesi için ortamın çeşitlendirilmesi gerekir. Bu noktada "domain randomization" adı verilen teknik devreye girer. Bu yöntemle simülasyon ortamındaki parametreler sürekli değiştirilir. Zemin sürtünmesi, nesne ağırlıkları, kamera açıları ve ışık koşulları rastgele varyasyonlarla sunulur. Böylece robot tek bir sabit dünyayı değil, birçok olası dünyayı öğrenir.

Simülasyon eğitiminde bir diğer önemli kavram sensör modellemesidir. Robotlar dünyayı insan gibi görmez; kameralar, lidarlar ve derinlik sensörleri üzerinden algılar. Ancak bu sensörler kusursuz değildir. Gürültü, eksik veri ve bozulmalar içerir. Isaac Sim, bu gerçekçiliği simüle ederek robotların yalnızca ideal değil, bozulmuş verilerle de çalışabilmesini sağlar. Bu, gerçek dünyaya geçişte kritik bir avantajdır.

Eğitim sürecinin merkezinde öğrenme algoritmaları bulunur. En yaygın yaklaşım pekiştirmeli öğrenmedir. Bu yöntemde robot bir durum gözlemler, bir eylem seçer ve ortamdan bir geri bildirim alır. Bu geri bildirim ödül fonksiyonu tarafından belirlenir. Örneğin bir nesneyi doğru şekilde kavrarsa pozitif ödül alır, düşürürse negatif ödül alır. Zamanla robot, maksimum ödül getiren stratejileri öğrenir.

Bu süreçte sinir ağları kritik rol oynar. Robotun karar verme mekanizması genellikle derin öğrenme modelleriyle temsil edilir. Bu modeller, yüksek boyutlu sensör verilerini işleyerek eylem üretir. Eğitim ilerledikçe sinir ağı, karmaşık davranışları otomatik olarak öğrenir. Örneğin bir robot kolu, başlangıçta rastgele hareket ederken zamanla hassas kavrama hareketleri geliştirebilir.

Simülasyonda eğitim yalnızca bireysel robotlar için değil, çoklu ajan sistemleri için de uygulanabilir. Aynı ortamda birden fazla robotun birlikte öğrenmesi, rekabet veya iş birliği davranışlarını ortaya çıkarır. Bu durum özellikle lojistik, üretim ve otonom araç sistemlerinde önemlidir. Robotlar yalnızca bireysel görevleri değil, kolektif davranışları da öğrenir.

Bununla birlikte simülasyon eğitiminin en büyük zorluklarından biri “sim-to-real gap”tir. Yani simülasyonda öğrenilen davranışların gerçek dünyaya tam olarak aktarılmaması sorunu. Bu fark, fiziksel modellerin mükemmel olmamasından kaynaklanır. Gerçek dünyada sürtünme değişken, sensörler gürültülü ve ortam sürekli dinamiktir. Bu farkı azaltmak için çeşitli teknikler kullanılır: gürültü ekleme, rastgelelik artırma ve model çeşitlendirme gibi.

Isaac Lab gibi platformlar bu geçiş sürecini kolaylaştırmak için özel araçlar sunar. Eğitim sırasında elde edilen modeller sürekli test edilir ve farklı senaryolara maruz bırakılır. Bu sayede robot yalnızca belirli bir ortamı değil, geniş bir dağılımı öğrenmiş olur. Bu yaklaşım, genelleme yeteneğini önemli ölçüde artırır.

Simülasyonda robot eğitimi aynı zamanda veri üretimi açısından da devrim niteliğindedir. Geleneksel yöntemlerde veri toplamak zor ve pahalıdır. Ancak simülasyon ortamında veri otomatik olarak üretilir. Her hareket, her etkileşim bir veri noktasıdır. Bu durum, büyük ölçekli yapay zekâ modellerinin eğitilmesini mümkün kılar.

Sonuç olarak simülasyonda robot eğitimi, robotik alanında paradigmatik bir dönüşüm yaratmıştır. Artık robotlar fiziksel dünyada değil, dijital bir evrende doğar ve gelişir. Bu süreç, daha güvenli, daha hızlı ve daha ölçeklenebilir bir öğrenme modeli sunar. Isaac Sim ve Isaac Lab gibi sistemler sayesinde robotlar yalnızca programlanan makineler değil, deneyim yoluyla öğrenen sistemler haline gelir. Bu da bizi klasik mühendislikten çıkarıp, öğrenen sistemler çağının merkezine yerleştirir.

5.4 Reinforcement Learning ile Davranış Öğretmek

Pekiştirmeli öğrenme (Reinforcement Learning - RL), robotlara doğrudan “ne yapmaları gerektiğini” söylemek yerine, “neyi başardıklarında ödül alacaklarını” tanımlayarak davranış öğretme yaklaşımıdır. Bu yaklaşımın temel fikri oldukça basittir: bir ajan (robot), bir ortam içinde hareket eder, yaptığı her eylemden sonra bir geri bildirim alır ve zaman içinde en yüksek toplam ödülü getiren davranışları öğrenir.

Bu bakış açısı, robotik sistemleri klasik yazılım dünyasından tamamen ayırır. Geleneksel programlamada mühendis “if-else” kuralları yazar; ancak RL’de mühendis yalnızca hedefi ve ödül yapısını tanımlar. Robot ise deneme-yanılma yoluyla kendi stratejisini geliştirir. Bu nedenle RL, özellikle Isaac Sim gibi simülasyon ortamlarında son derece güçlü bir araçtır.
Temel bileşenler üçtür:

- Ajan (Agent): Öğrenen sistem, yani robot
- Ortam (Environment): Robotun etkileşimde bulunduğu dünya
- Ödül (Reward): Robotun davranışını değerlendiren sinyal

Bu sistem sürekli bir döngü içinde çalışır:

1. Ajan ortamı gözlemler
2. Bir eylem seçer
3. Ortam yeni duruma geçer
4. Ödül veya ceza verilir
5. Ajan politikayı günceller

Bu döngü milyonlarca kez tekrarlandığında, ajan giderek daha iyi davranışlar geliştirir.
Aşağıda bu süreci anlamak için basit ama öğretici bir örnek kuracağız: ızgara tabanlı (grid world) bir ortamda hedefe ulaşmaya çalışan bir ajan.

Basit Problem: Grid World

Diyelim ki 4x4 bir kare ızgaramız var. Ajan (robot) sol üst köşeden başlıyor ve sağ alt köşedeki hedefe ulaşmak istiyor. Kurallar:

- Ajan yukarı, aşağı, sağ, sol hareket edebilir
- Hedefe ulaşınca +10 ödül alır
- Her adım -1 ceza getirir (gereksiz dolaşmayı engeller)
- Duvara çarparsa yerinde kalır

Amaç: En kısa yolu öğrenmek.

Ortamın Tanımı (Environment)

Aşağıdaki Python kodu basit bir RL ortamını tanımlar:

```
import numpy as np
import random

class GridWorld:
    def __init__(self):
        self.size = 4
        self.start = (0, 0)
        self.goal = (3, 3)
        self.state = self.start

    def reset(self):
        self.state = self.start
        return self.state

    def step(self, action):
        x, y = self.state

        # 0: up, 1: down, 2: left, 3: right
        if action == 0:
            x = max(0, x - 1)
        elif action == 1:
            x = min(self.size - 1, x + 1)
        elif action == 2:
            y = max(0, y - 1)
        elif action == 3:
            y = min(self.size - 1, y + 1)

        self.state = (x, y)

        reward = -1
        done = False

        if self.state == self.goal:
            reward = 10
            done = True

        return self.state, reward, done
```

Bu ortam, gerçek robotik sistemlerin çok basitleştirilmiş bir modelidir. Örneğin Isaac Sim içinde bu “grid” yerine fiziksel dünya, sensörler ve çarpışmalar vardır. Ancak öğrenme mantığı aynıdır.

Q-Learning ile Öğrenme

Şimdi robotun nasıl öğrendiğini tanımlayalım. En temel RL algoritmalarından biri Q-learning’dir. Q-learning’in mantığı şudur: Her durum-eylem çiftine bir “değer” atanır. Bu değer, o eylemin gelecekte ne kadar iyi sonuç vereceğini temsil eder.

Q-learning güncellemesi:

$Q(s, a) \leftarrow Q(s, a) + \alpha [\text{ödül} + \gamma \max Q(s', a') - Q(s, a)]$
Burada:

- α: öğrenme oranı
- γ: geleceğe verilen önem

Bu formül, robotun deneyimlerinden öğrenmesini sağlar.

Q-Tablosu ile Basit RL Ajanı

```
class QLearningAgent:
    def __init__(self, actions):
        self.q_table = {}
        self.actions = actions
        self.alpha = 0.1
        self.gamma = 0.9
        self.epsilon = 0.2  # keşif oranı

    def get_q(self, state, action):
        return self.q_table.get((state, action), 0.0)

    def choose_action(self, state):
        if random.random() < self.epsilon:
            return random.choice(self.actions)

        q_values = [self.get_q(state, a) for a in self.actions]
        return self.actions[np.argmax(q_values)]

    def learn(self, state, action, reward, next_state):
        old_q = self.get_q(state, action)
        future_q = max([self.get_q(next_state, a) for a in self.actions])

        new_q = old_q + self.alpha * (reward + self.gamma * future_q - old_q)
        self.q_table[(state, action)] = new_q
```

Eğitimi Başlatma

Şimdi ajanı eğitelim:

```
env = GridWorld()
agent = QLearningAgent(actions=[0, 1, 2, 3])

episodes = 500

for episode in range(episodes):
    state = env.reset()
```

```
    done = False

    while not done:
        action = agent.choose_action(state)
        next_state, reward, done = env.step(action)

        agent.learn(state, action, reward, next_state)

        state = next_state

print("Eğitim tamamlandı.")
```

Öğrenme Süreci Ne Anlama Geliyor?

Başlangıçta ajan tamamen rastgele hareket eder. Hedefe ulaşması tesadüftür. Ancak zamanla:

- Hedefe götüren yollar daha yüksek Q değerleri alır
- Yanlış hareketler cezalandırılır
- Ajan giderek en kısa yolu öğrenir

Bu süreç, robotik sistemlerde çok daha karmaşıktır ama temel mantık aynıdır.

Gerçek Robotlara Bağlantı

Bu basit örnek, aslında gerçek robotik sistemlerin küçük bir modelidir. Gerçek dünyada:

- Durum = sensör verileri (kamera, lidar)
- Eylem = motor komutları
- Ödül = görev başarısı (örneğin nesneyi kavrama)

Örneğin bir robot kolu düşünelim. Görevi bir nesneyi tutmak olsun:

- Nesne tutulursa +100
- Düşürülürse -50
- Gereksiz hareketler -1

Robot binlerce denemeden sonra en stabil kavrama stratejisini öğrenir.

Isaac Sim ile RL

Gerçek sistemlerde bu süreç NVIDIA Isaac Sim içinde çalıştırılır. Burada:

- Fizik motoru (örneğin NVIDIA PhysX) gerçek dünya dinamiklerini simüle eder
- Sensörler gerçekçi veri üretir
- RL ajanı bu ortamda eğitim alır

Bu sayede robotlar gerçek dünyaya çıkmadan önce “deneyim kazanır”.

Neden RL Bu Kadar Güçlü?

Pekiştirmeli öğrenmenin en önemli avantajı şudur:
Robotu programlamazsınız, onu yetiştirirsiniz.
Bu yaklaşım sayesinde:

- Karmaşık kontrol problemleri otomatik çözülür
- İnsan sezgisi gerektiren stratejiler ortaya çıkar
- Sistem kendi optimizasyon yolunu bulur

Örneğin:

- Bir robot yürümeyi öğrenebilir
- Bir drone rüzgarda dengede kalabilir
- Bir robot kol hassas montaj yapabilir

Keşif ve Sömürü Dengesi

RL'de kritik bir problem vardır: exploration vs exploitation.

- Exploration: Yeni şeyler denemek
- Exploitation: Bildiği en iyi şeyi yapmak

Eğer robot sadece bildiğini yaparsa yeni stratejiler öğrenemez. Ama sürekli rastgele denerse de asla optimize olamaz. Bu denge RL'nin kalbidir.
Epsilon-greedy yöntemi bu yüzden kullanılır:

- %20 rastgele hareket
- %80 en iyi bilinen hareket

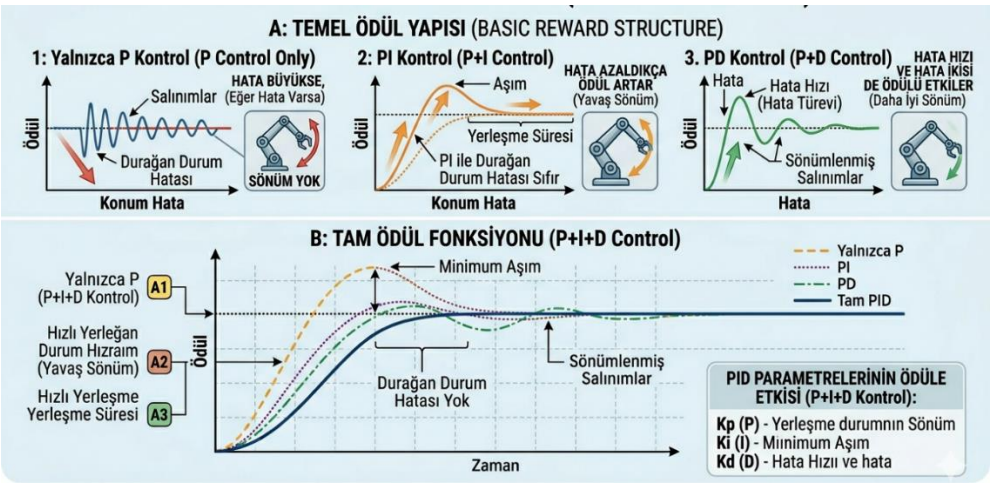
Sonuç

Pekiştirmeli öğrenme, robotlara davranış öğretmenin en güçlü yollarından biridir. Basit bir grid world örneğinden başlayıp Isaac Sim gibi gelişmiş simülasyonlara kadar uzanan bu yaklaşım, modern robotik sistemlerin temelini oluşturur.
Bu yöntemde mühendis artık bir "komut yazarı" değil, bir "öğrenme mimarıdır". Robot ise pasif bir makine değil, deneyimle gelişen bir sistemdir.
Bu dönüşüm, robotik alanını klasik mühendislikten çıkarıp öğrenen sistemler çağının merkezine taşımaktadır.

5.5 Reward Function: Robotu Ne Motive Eder?

İzleyen Şekil 10'daki ödül fonksiyonu görseli, klasik kontrol teorisindeki PID bileşenlerinin pekiştirmeli öğrenme (RL) bağlamındaki ödül yapısı (Reward Structure) ile nasıl ilişkilendirildiğini özetlemektedir. Şema, bir robot kolu örneği üzerinden hata payı ve ödül arasındaki dengeyi üç ana aşamada ele alır.

Şekil 10. Ödül Fonksiyonu



Pekiştirmeli öğrenmede en kritik unsur çoğu zaman algoritma ya da model değil, “ödül fonksiyonu”dur. Çünkü bir robotun nasıl davranacağını belirleyen şey, doğrudan doğruya neyi “iyi” ve neyi “kötü” olarak öğrendiğidir. Başka bir deyişle, robotun motivasyonu matematiksel olarak tanımlanır. Bu motivasyon yanlış tasarlanırsa, en güçlü öğrenme algoritması bile hatalı davranışlar üretir.
Ödül fonksiyonu, robotun hedefini sayısallaştırır. İnsan için “bir nesneyi dikkatli şekilde kaldırmak” sezgisel bir kavramdır. Ancak robot için bu, sayılara çevrilmek zorundadır: başarı +10, düşürme -50, zaman kaybı -1 gibi.

Bu bölümde basit ama öğretici bir örnek üzerinden ilerleyeceğiz: bir robot kolunun bir nesneyi alıp hedef bölgeye taşıması.

Problem: Nesne Taşıyan Robot

Bir robot kolu düşünelim. Görev:

- Masadaki bir nesneyi al
- Hedef kutuya bırak

Bu görevde robotun davranışını yönlendiren şey reward function’dır.

Reward Function Neyi Belirler?

Bir ödül fonksiyonu şunları tanımlar:

- Robot ne zaman başarılı sayılır?
- Hangi hatalar cezalandırılır?
- Hangi davranışlar teşvik edilir?

Yanlış tasarlanmış bir reward function şu sorunlara yol açabilir:

- Robot hedefe gitmeden “ödül hack’leyebilir”
- Gereksiz hareketlerle ödül toplayabilir
- Fiziksel olarak anlamsız stratejiler geliştirebilir

Bu nedenle reward function, robotun “ahlak sistemi” gibidir.

Basit Bir Robotik Simülasyon

Aşağıdaki örnekte 2D bir robot kolu var:

- Amaç: (0,0)’dan bir nesneyi alıp (5,5)’e götürmek
- Robot her adımda hareket ediyor

Ortam Tanımı

```
import numpy as np

class SimpleRobotEnv:
    def __init__(self):
        self.reset()

    def reset(self):
        self.robot_pos = np.array([0.0, 0.0])
        self.object_pos = np.array([2.0, 2.0])
        self.target_pos = np.array([5.0, 5.0])
        self.holding = False
        return self.get_state()

    def get_state(self):
        return np.concatenate([self.robot_pos, self.object_pos])

    def step(self, action):
        # action: [dx, dy]
        self.robot_pos += action

        reward = self.compute_reward()
        done = self.check_done()

        return self.get_state(), reward, done

    def check_done(self):
        return np.linalg.norm(self.robot_pos - self.target_pos) < 0.5

    def compute_reward(self):
        reward = 0

        # Nesneye yaklaşma ödülü
        dist_to_object = np.linalg.norm(self.robot_pos - self.object_pos)
        reward += -dist_to_object * 0.1

        # Nesneyi tutma bonusu
        if dist_to_object < 0.5:
            reward += 5
            self.holding = True

        # Hedefe ulaşma ödülü
        if self.holding:
            dist_to_target = np.linalg.norm(self.robot_pos - self.target_pos)
            reward += -dist_to_target * 0.2

        # Başarı ödülü
        if np.linalg.norm(self.robot_pos - self.target_pos) < 0.5:
```

```
            reward += 20

        return reward
```

Bu Reward Function Ne Yapıyor?

Bu fonksiyon aslında robotun “motivasyon haritasını” çiziyor:

1. Nesneye yaklaş → küçük ödül
2. Nesneyi yakala → bonus
3. Hedefe yaklaş → ek ödül
4. Başarı → büyük ödül

Aynı zamanda:

- Uzaklaşma → ceza
- Zaman kaybı → dolaylı ceza

Bu yapı, robotu doğal olarak doğru davranışa iter.

Basit Bir Ajan (Random Policy)

Şimdi robotun bu ortamda nasıl davrandığını görelim:

```
import random

env = SimpleRobotEnv()

state = env.reset()
total_reward = 0

for step in range(50):
    # Basit rastgele hareket
    action = np.random.uniform(-1, 1, size=2)

    state, reward, done = env.step(action)

    total_reward += reward

    print(f"Step {step}, Reward: {reward:.2f}")

    if done:
        print("Görev tamamlandı!")
        break

print("Toplam ödül:", total_reward)
```

Bu kodda ajan öğrenmiyor; sadece reward function'ın nasıl çalıştığını gözlemliyoruz.

Reward Function Olmadan Ne Olur?

Eğer reward function olmasaydı:

- Robot hiçbir yön seçemezdi
- Rastgele hareket ederdi
- Öğrenme gerçekleşmezdi

Yani reward function yoksa "öğrenme problemi" de yoktur.

Kötü Tasarlanmış Reward Örneği

Şimdi hatalı bir reward function düşünelim:

```
def bad_reward(self):
  reward = 0

  # Sadece hareket etmek ödül
  reward += 1

  # Hedefe bakmadan sürekli dolaşmak daha iyi olur
  return reward
```

Bu durumda robot şunu öğrenir:

- Sürekli hareket et → ödül kazan
- Hedefe gitmek gereksiz

Sonuç: Robot "amaçsız gezici" olur.

Reward Hacking Problemi

Gerçek sistemlerde robotlar bazen reward function'ı "hack'ler". Örneğin:

- Bir robot temizlik yapması gerekirken kirleri köşeye iter
- Bir drone görev alanında dönüp durarak ödül toplar

Bu durum, yanlış tanımlanmış motivasyon sisteminin sonucudur.

Isaac Sim ile Reward Design

Gerçek dünyada bu süreç NVIDIA Isaac Sim içinde çalıştırılır. Burada reward function:

- Sensör verilerine bağlanır
- Fizik motoru (NVIDIA PhysX) ile etkileşir

- Gerçekçi ortam koşullarına göre hesaplanır

Örneğin:

- Çarpma → büyük ceza
- Stabil tutuş → küçük ödül
- Enerji tüketimi → ek ceza

Reward Function Tasarım Prensipleri

İyi bir reward function şu özellikleri taşır:

1. **Net olmalı**: Belirsiz hedefler olmamalı
2. **Dengeli olmalı**: Aşırı ödül veya ceza olmamalı
3. **Hedef odaklı olmalı**: Yan davranışları teşvik etmemeli
4. **Ölçeklenebilir olmalı**: Farklı durumlara genellenmeli

Sezgisel Bir Analoji

Reward function, aslında robotun "iç motivasyon sistemi" gibidir.
İnsan için:

- Açlık → yemek yeme davranışı
- Sosyal ödül → konuşma davranışı
- Başarı hissi → öğrenme motivasyonu

Robot için:

- Sayısal ödül → davranış seçimi

Yani robotların "istekleri" yoktur, sadece optimize ettikleri fonksiyonlar vardır.

Sonuç

Reward function, pekiştirmeli öğrenmenin kalbidir. Yanlış tanımlanırsa robot yanlış şeyleri öğrenir; doğru tanımlanırsa karmaşık davranışlar kendiliğinden ortaya çıkar.
Bu nedenle robotik sistemlerde asıl mühendislik problemi artık mekanik tasarım değil, "motivasyon tasarımıdır". Isaac Sim ve modern RL sistemlerinde başarı, çoğu zaman en iyi modeli yazmakla değil, en doğru reward fonksiyonunu tasarlamakla elde edilir.

5.6 Paralel Öğrenme ve Hızlandırılmış Eğitim

Robotik öğrenmenin en büyük darboğazlarından biri, gerçek dünyada deneyim toplamanın yavaş ve pahalı olmasıdır. Bir robot kolunun tek bir hareketi bile milisaniyeler sürerken, anlamlı bir beceri öğrenmesi için milyonlarca deneme gerekir. Bu ölçek, fiziksel dünyada pratik olarak imkânsızdır. İşte bu noktada paralel öğrenme ve hızlandırılmış eğitim kavramı devreye girer ve modern robotik sistemlerin ölçeğini tamamen değiştirir.

Isaac Lab ve NVIDIA Isaac Sim gibi platformların en güçlü yönlerinden biri, aynı anda binlerce hatta milyonlarca simülasyon ortamını paralel olarak çalıştırabilmesidir. Bu yaklaşım, tek bir robotun öğrenmesini beklemek yerine, aynı anda "binlerce robot kopyasının" farklı deneyimler yaşamasını sağlar. Böylece zaman, doğrusal olmaktan çıkar ve adeta sıkıştırılmış bir öğrenme sürecine dönüşür.

Paralel Öğrenme Nedir?

Paralel öğrenme, aynı öğrenme algoritmasının çok sayıda bağımsız ortamda eşzamanlı olarak çalıştırılmasıdır. Her ortam farklı bir deneyim üretir, ancak tüm deneyimler aynı model ağına katkı yapar.

Basitçe düşünelim:

- Tek robot → 1 deneyim / saniye
- 10.000 robot → 10.000 deneyim / saniye

Bu fark, öğrenme hızını dramatik biçimde artırır.

Neden Paralel Öğrenmeye İhtiyaç Var?

Robotik sistemlerde öğrenme maliyeti şu üç nedenle yüksektir:

1. **Fiziksel etkileşim yavaştır**
2. **Deneyim toplamak pahalıdır**
3. **Gerçek dünyada hata risklidir**

Örneğin bir robot kolunun kavrama becerisi için:

- 1 milyon deneme gerekiyorsa
- Gerçek dünyada bu yıllar sürebilir

Paralel simülasyon ise bu süreci saatler veya günler seviyesine indirir.

Isaac Sim'de Paralel Ortamlar

Isaac Sim, GPU tabanlı fizik motorları sayesinde aynı anda çok sayıda ortamı çalıştırabilir. Her ortam:

- Bağımsız fizik kurallarına sahiptir
- Farklı başlangıç koşulları içerir
- Aynı öğrenme ajanına veri sağlar

Bu yapı, özellikle GPU mimarisinin doğal paralelliğinden yararlanır.

Basit Bir Paralel Eğitim Mantığı

Aşağıdaki örnek, kavramsal olarak paralel öğrenmenin nasıl çalıştığını gösterir:

```
import numpy as np

class ParallelEnv:
    def __init__(self, n_envs=100):
        self.n_envs = n_envs
        self.positions = np.zeros((n_envs, 2))
        self.targets = np.random.uniform(-5, 5, (n_envs, 2))

    def step(self, actions):
        # actions: (n_envs, 2)
        self.positions += actions

        rewards = -np.linalg.norm(self.positions - self.targets, axis=1)

        done = np.linalg.norm(self.positions - self.targets, axis=1) < 0.5

        return self.positions, rewards, done
```

Bu kodda aynı anda 100 farklı robot ortamı çalışmaktadır.

Paralel Öğrenme Ajanı

Şimdi bu ortamdan veri toplayan basit bir ajan düşünelim:

```
class ParallelAgent:
    def __init__(self, n_envs):
        self.n_envs = n_envs

    def act(self, states):
        # Basit rastgele politika
        return np.random.uniform(-1, 1, (self.n_envs, 2))
```

Eğitim Döngüsü

```
env = ParallelEnv(n_envs=1000)
agent = ParallelAgent(n_envs=1000)

for step in range(200):
    actions = agent.act(env.positions)

    states, rewards, done = env.step(actions)

    avg_reward = np.mean(rewards)
```

```
    if step % 10 == 0:
        print(f"Step {step}, Ortalama ödül: {avg_reward:.3f}")
```

Bu yapı, binlerce robotun aynı anda öğrenmesini simüle eder.

Paralel Öğrenmenin Gücü

Paralel öğrenmenin en önemli avantajları:

1. Veri üretimi hızlanır
Tek bir robot yerine binlerce robot veri üretir.
2. Genelleme artar
Farklı ortamlarda öğrenme, modeli daha dayanıklı hale getirir.
3. Eğitim süresi kısalır
Günler süren eğitimler saatlere düşebilir.
4. Keşif kapasitesi artar
Farklı ajanlar farklı stratejiler keşfeder.

Asenkron Öğrenme

Paralel sistemlerde tüm ortamlar senkron olmak zorunda değildir. Asenkron öğrenme yaklaşımında:

- Her ortam farklı hızda ilerler
- Deneyimler merkezi modele gönderilir
- Model sürekli güncellenir

Bu yapı özellikle büyük ölçekli sistemlerde kullanılır.

GPU Tabanlı Hızlandırma

Paralel öğrenmenin gerçek gücü GPU mimarisi ile ortaya çıkar. CPU tabanlı sistemlerde paralellik sınırlıdır, ancak GPU'lar binlerce işlemi aynı anda yapabilir. NVIDIA PhysX gibi fizik motorları ve Isaac Sim altyapısı, bu GPU paralelliğini kullanarak fizik hesaplamalarını hızlandırır. Bu sayede:

- Çarpışmalar
- Sürtünme hesapları
- Sensör simülasyonları

eşzamanlı olarak işlenir.

Domain Randomization ile Birleşim

Paralel öğrenme genellikle domain randomization ile birlikte kullanılır. Her paralel ortam farklıdır:

- Farklı ışık koşulları
- Farklı nesne konumları
- Farklı fizik parametreleri

Bu sayede model tek bir dünyaya değil, dağılımsal bir dünyaya uyum sağlar.

Gerçek Dünya ile Bağlantı

Paralel öğrenme sayesinde robotlar:

- Daha hızlı öğrenir
- Daha az veriyle genelleme yapabilir
- Gerçek dünyaya daha iyi transfer olur

Örneğin bir fabrika robotu:

- Tek bir ortamda değil
- 500 farklı sanal fabrika düzeninde eğitilir

Zihinsel Model: "Robot Ordusu"

Paralel öğrenmeyi anlamanın en iyi yolu şudur:

Tek bir robot değil, binlerce robotluk bir ordu düşünün.

- Her robot farklı bir deneyim yaşar
- Hepsi aynı beyne katkı yapar
- Kolektif olarak öğrenme gerçekleşir

Bu, biyolojik evrim mantığına da benzer: birçok varyasyon → doğal seçim → en iyi davranış.

Sonuç

Paralel öğrenme ve hızlandırılmış eğitim, modern robotik sistemlerin ölçeklenmesini mümkün kılan en kritik teknolojilerden biridir. Isaac ekosistemi sayesinde robotlar artık tek tek değil, toplu halde öğrenir. Bu da öğrenme sürecini doğrusal olmaktan çıkarıp üstel bir yapıya dönüştürür. Sonuç olarak robotik artık bireysel makinelerin değil, paralel öğrenen sistemlerin alanıdır. Bu dönüşüm, yapay zekânın fiziksel dünyaya entegrasyonunda en önemli kırılma noktalarından biridir.

5.7 Sim-to-Real: Simülasyondan Gerçek Dünyaya Geçiş

Robotik öğrenmenin en kritik ve en zor aşaması, simülasyonda öğrenilen davranışların gerçek dünyaya aktarılmasıdır. Bu problem "sim-to-real transfer" olarak adlandırılır ve modern robotik sistemlerin başarısını belirleyen temel eşiklerden biridir. Çünkü bir robot simülasyonda mükemmel performans gösterse bile, aynı davranışı gerçek dünyada sergileyemeyebilir. Bunun

nedeni, simülasyonun her ne kadar gelişmiş olursa olsun, fiziksel dünyanın tüm karmaşıklığını birebir yansıtamamasıdır.

Isaac Lab ve NVIDIA Isaac Sim gibi platformlar bu problemi çözmek için özel olarak tasarlanmıştır. Bu sistemler, yalnızca robotları eğitmekle kalmaz; aynı zamanda simülasyon ile gerçek dünya arasındaki farkı minimize etmeye çalışır. Bu farkı anlamak, modern robotik mühendisliğinin merkezinde yer alır.

Simülasyon Neden Yeterli Değildir?

Simülasyonlar fizik kurallarını matematiksel modellerle temsil eder. Ancak gerçek dünya:

- Tamamen deterministik değildir
- Sürekli gürültü içerir
- Beklenmeyen etkileşimler üretir

Örneğin:

- Bir nesnenin yüzeyi simülasyonda pürüzsüz olabilir
- Gerçekte ise mikro düzeyde düzensizlikler vardır
- Simülasyonda sensörler kusursuz çalışır
- Gerçekte kamera görüntüsü ışık, gölge ve gürültü içerir

Bu küçük farklar, robot davranışını ciddi şekilde etkileyebilir.

Sim-to-Real Gap Nedir?

Sim-to-real gap, simülasyon ortamı ile gerçek dünya arasındaki davranış farkıdır. Bu fark üç ana kaynaktan oluşur:

1. **Fiziksel model hataları**
2. **Sensör gürültüsü**
3. **Kontrol gecikmeleri**
4.

Bu fark büyüdükçe, simülasyonda öğrenilen politika gerçek dünyada başarısız olur.

Domain Randomization: Kaosu Öğrenmek

Sim-to-real problemini çözmek için en yaygın yöntemlerden biri domain randomization'dır. Bu yaklaşımın temel fikri şudur:

Robotu tek bir "mükemmel dünyada" eğitmek yerine, birçok farklı "rastgele dünyada" eğitmek. Örneğin:

- Zemin sürtünmesi sürekli değişir
- Nesne ağırlıkları rastgele seçilir
- Işık koşulları farklılaştırılır
- Kamera açısı sürekli değiştirilir

Bu sayede robot, belirli bir ortama değil, olasılıklar dağılımına uyum sağlar.

Basit Sim-to-Real Randomization Örneği

Aşağıdaki örnek, basit bir simülasyon ortamında rastgelelik eklenmesini gösterir:

```
import numpy as np

class RandomizedEnvironment:
    def __init__(self):
        self.reset()

    def reset(self):
        self.friction = np.random.uniform(0.2, 1.0)
        self.object_mass = np.random.uniform(0.5, 2.0)
        self.noise_level = np.random.uniform(0.0, 0.1)

        self.position = np.array([0.0, 0.0])
        return self.position

    def step(self, action):
        noise = np.random.normal(0, self.noise_level, size=2)

        # Basit fizik modeli
        self.position += action * self.friction / self.object_mass + noise

        reward = -np.linalg.norm(self.position – np.array([5.0, 5.0]))

        done = np.linalg.norm(self.position – np.array([5.0, 5.0])) < 0.5

        return self.position, reward, done
```

Bu yapı sayesinde robot her denemede farklı bir dünyada öğrenir.

Sensör Gerçekçiliği

Gerçek dünyada robotlar mükemmel veri almaz. Isaac Sim gibi sistemler, sensörleri gerçekçi şekilde modelleyerek bu farkı azaltır.

Örneğin:

- Kamera görüntülerine bulanıklık eklenir
- Lidar ölçümlerine gürültü katılır
- Gecikmeler simüle edilir

Bu yaklaşım, robotun “kusurlu veriyle çalışmayı öğrenmesini” sağlar.

Fiziksel Parametre Belirsizliği

Gerçek dünyada fiziksel parametreler sabit değildir:

- Sürtünme değişebilir
- Nesne ağırlıkları tam bilinmez
- Motor tepkileri farklılık gösterebilir

Bu nedenle simülasyonda bu parametreler sabit tutulmaz. Bunun yerine aralıklar içinde rastgele seçilir. NVIDIA PhysX bu fiziksel değişkenleri gerçek zamanlı olarak modelleyerek simülasyonun çeşitliliğini artırır.

Öğrenilen Politikanın Transferi

Simülasyonda eğitilen bir model genellikle bir "politika ağı"dır. Bu ağ:

- Sensör verisini alır
- Eylem üretir

Transfer sürecinde bu ağ doğrudan gerçek robota yüklenir. Ancak çoğu zaman ek bir kalibrasyon gerekir.

Fine-Tuning: Gerçek Dünya Ayarı

Sim-to-real transfer sonrası genellikle küçük bir "fine-tuning" süreci yapılır:

- Robot gerçek dünyada birkaç deneme yapar
- Model bu verilerle güncellenir
- Performans stabilize edilir

Bu adım, simülasyon ile gerçek dünya arasındaki son boşluğu kapatır.

Dijital İkizler (Digital Twin)

Sim-to-real yaklaşımının en güçlü araçlarından biri dijital ikiz teknolojisidir. Gerçek sistemin birebir dijital kopyası oluşturulur. Bu sayede:

- Gerçek fabrikalar simülasyonda test edilir
- Robot davranışları önceden doğrulanır
- Riskli senaryolar güvenli şekilde analiz edilir

Sim-to-Real Başarısının Anahtarı

Başarılı bir transfer için üç temel unsur gerekir:

1. **Yüksek kaliteli simülasyon**

2. **Yeterli rastgelelik (randomization)**
3. **Genelleme kapasitesi yüksek modeller**

Bu üçlü birlikte çalışmadığında transfer başarısız olur.

Neden Bu Problem Zor?

Sim-to-real problemi zordur çünkü:

- Gerçek dünya sonsuz değişken içerir
- Simülasyon her zaman bir yaklaşımdır
- Küçük hatalar büyük davranış farkları yaratır

Örneğin:

- Simülasyonda %1 hata → gerçek dünyada tamamen farklı davranış

Isaac Ekosisteminde Sim-to-Real

Isaac ekosistemi bu problemi azaltmak için entegre bir yapı sunar:

- Isaac Sim → gerçekçi simülasyon ortamı
- Isaac Lab → öğrenme altyapısı
- PhysX → fizik doğruluğu
-

Bu üçlü birlikte çalışarak transfer başarısını artırır.

Sonuç

Sim-to-real problemi, robotik öğrenmenin en kritik sınavıdır. Bir modelin başarılı olup olmadığı, simülasyonda değil, gerçek dünyada belli olur. Bu nedenle modern robotik sistemler yalnızca öğrenmeye değil, "dayanıklılığa" da odaklanır.

Bugün geldiğimiz noktada hedef, sadece simülasyonda iyi çalışan robotlar değil, gerçek dünyada güvenilir şekilde davranabilen sistemler üretmektir. Sim-to-real köprüsü, bu hedefe giden en önemli geçittir.

5.8 Domain Randomization ve Dayanıklı Davranış

Robotik öğrenmede en temel sorunlardan biri şudur: Simülasyonda mükemmel çalışan bir model, gerçek dünyaya geçtiğinde neden başarısız olur? Bu sorunun cevabı genellikle tek bir noktada toplanır: dünya sabit değildir. Fizik değişir, ışık değişir, sensörler değişir, hatta aynı nesne bile farklı ortamlarda farklı davranır. İşte domain randomization yaklaşımı tam da bu değişkenliği öğrenme sürecinin merkezine yerleştirir.

Isaac Lab ve NVIDIA Isaac Sim gibi modern robotik platformlar, bu yaklaşımı simülasyonun doğal bir parçası haline getirir. Amaç, robotu tek bir "ideal dünya"ya değil, olası dünyaların

dağılımına karşı eğitmektir. Böylece robot, belirli bir ortamı ezberlemez; bunun yerine değişkenlik altında da çalışabilen dayanıklı (robust) davranışlar geliştirir.

Domain Randomization Nedir?

Domain randomization, simülasyon ortamındaki tüm değişkenlerin rastgeleleştirilmesi yaklaşımıdır. Buradaki "domain" kelimesi, robotun öğrendiği dünya anlamına gelir. "Randomization" ise bu dünyanın sürekli değiştirilmesidir. Bu yaklaşımda:

- Nesnelerin boyutları değişir
- Yüzeylerin sürtünmesi farklılaşır
- Işık kaynakları rastgele konumlanır
- Kamera açıları sürekli değişir
- Sensör gürültüsü dinamik olarak eklenir

Robot, tek bir sabit gerçeklik yerine, sonsuz varyasyonlar içinde öğrenir.

Neden Bu Kadar Önemli?

Gerçek dünya deterministik değildir. Aynı görev:

- Sabah ve akşam farklı ışıkta yapılır
- Farklı yüzeylerde farklı sürtünme üretir
- Farklı cihazlarda farklı sensör verisi oluşturur

Eğer robot sadece tek bir simülasyon ortamında eğitilirse, bu değişkenlikler karşısında kırılgan olur. Domain randomization bu kırılganlığı ortadan kaldırır.

Basit Bir Domain Randomization Örneği

Aşağıdaki Python örneği, bir robotik simülasyonda ortamın nasıl rastgeleleştirildiğini gösterir:

```
import numpy as np

class RandomizedRobotWorld:
    def __init__(self):
        self.reset()

    def reset(self):
        # Fiziksel değişkenler
        self.friction = np.random.uniform(0.1, 1.5)
        self.mass = np.random.uniform(0.5, 3.0)

        # Görsel değişkenler
        self.light_intensity = np.random.uniform(0.2, 2.0)
        self.camera_angle = np.random.uniform(-30, 30)
```

```
        # Sensör gürültüsü
        self.sensor_noise = np.random.uniform(0.0, 0.2)

        self.state = np.array([0.0, 0.0])
        return self.state

    def step(self, action):
        noise = np.random.normal(0, self.sensor_noise, size=2)

        # Basit dinamik model
        self.state += action * self.friction / self.mass + noise

        reward = -np.linalg.norm(self.state – np.array([5.0, 5.0]))

        done = np.linalg.norm(self.state – np.array([5.0, 5.0])) < 0.5

        return self.state, reward, done
```

Bu ortamda robot her episode'da farklı bir fizik ve algı dünyasında yaşar.

Dayanıklı Davranış (Robust Behavior) Nedir?

Domain randomization'ın amacı sadece çeşitlilik yaratmak değildir. Asıl hedef, "dayanıklı davranış" üretmektir. Dayanıklı davranış şu anlama gelir:

- Küçük değişiklikler performansı bozmaz
- Gürültü altında sistem çalışmaya devam eder
- Yeni ortamlara genelleme yapılabilir

Bu, robotun "tek bir doğruya" değil, "çok sayıda doğruya" uyum sağlamasıdır.

Öğrenme Sürecinde Etkisi

Domain randomization kullanıldığında öğrenme süreci şu şekilde değişir:

Normal eğitim:
- Tek ortam → yüksek performans
- Yeni ortam → başarısızlık

Randomized eğitim:
- Çoklu ortam → orta ama stabil performans
- Yeni ortam → kabul edilebilir performans

Yani amaç maksimum performans değil, güvenilir performanstır.

Görsel Domain Randomization

Robotların algı sistemleri için görsel randomization çok kritiktir. Örneğin:

- Nesneler farklı renklerde görünür
- Arka plan sürekli değişir
- Işık gölgeleri farklılaşır

Bu sayede robot “görsel ezber” yapmaz.

Fiziksel Domain Randomization

Fiziksel dünyadaki değişkenlikler:

- Sürtünme katsayısı
- Kütle dağılımı
- Elastikiyet
- Çarpışma tepkileri

Bu parametrelerin rastgeleleştirilmesi, robotun fiziksel sezgisini güçlendirir. NVIDIA PhysX bu tür fiziksel varyasyonları gerçek zamanlı olarak simüle edebilir.

Sim-to-Real Bağlantısı

Domain randomization, sim-to-real probleminin en etkili çözümlerinden biridir. Çünkü robot:

- Tek bir simülasyonu değil
- Simülasyonların dağılımını öğrenir

Bu nedenle gerçek dünya, bu dağılımın sadece bir örneği haline gelir.

Sezgisel Analoji

Domain randomization’ı anlamanın en iyi yolu insan öğrenmesine bakmaktır. Bir insan:

- Farklı ışıklarda görmeyi öğrenir
- Farklı seslerde konuşmayı anlar
- Farklı ortamlarda yürümeyi başarır

Çünkü beyin tek bir ortamda değil, değişken dünyada eğitilmiştir. Robotlar için domain randomization aynı işlevi görür.

Isaac Ekosisteminde Kullanımı

Isaac Lab içinde domain randomization şu alanlarda kullanılır:

- Robotik manipülasyon
- Navigasyon görevleri
- Görsel algı sistemleri
- Çok ajanlı sistemler

Bu sistemler, her eğitim episode'unda farklı bir dünya üretir.

Dayanıklılık vs Performans Dengesi

Domain randomization önemli bir trade-off yaratır:

- Performans biraz düşebilir
- Ama genelleme kapasitesi artar

Bu nedenle amaç "en iyi skor" değil, "en stabil sistem" üretmektir.

Gerçek Dünya Örneği

Bir depo robotu düşünelim:

- Yerde farklı sürtünme alanları var
- Kutular farklı ağırlıklarda
- Işık sürekli değişiyor

Eğer robot sadece tek bir ortamda eğitilirse başarısız olur. Ama domain randomization ile eğitilirse:

- Her koşula adapte olur
- Beklenmeyen durumlarda bile çalışır

Sonuç

Domain randomization, modern robotik öğrenmenin temel taşlarından biridir. Bu yaklaşım, robotları tek bir dünyaya bağımlı olmaktan çıkarır ve onları belirsizlik altında çalışan sistemlere dönüştürür. Sonuç olarak amaç, mükemmel bir dünyada mükemmel çalışan robotlar değil, kusurlu bir dünyada güvenilir şekilde çalışan robotlar üretmektir. Bu da dayanıklı davranışın temel tanımıdır.

5.9 Humanoid Robotlar Nasıl Eğitiliyor?

İnsana benzeyen robotlar, yani humanoid robotlar, robotik öğrenmenin en zor ama aynı zamanda en etkileyici alanını temsil eder. Çünkü burada mesele yalnızca bir görevi yerine getirmek değildir; denge kurmak, düşmemek, yürümek, çevreyle etkileşmek ve sürekli değişen fiziksel koşullara uyum sağlamak gibi çok sayıda karmaşık becerinin aynı anda öğrenilmesi gerekir. Bu nedenle humanoid robotların eğitimi, klasik robotik sistemlere kıyasla çok daha yüksek boyutlu bir problem uzayında gerçekleşir.

Bu tür sistemlerin eğitiminde Isaac Lab ve NVIDIA Isaac Sim gibi gelişmiş simülasyon ortamları kritik rol oynar. Çünkü gerçek bir humanoid robotu düşürerek öğrenmesini sağlamak hem tehlikeli hem de maliyetlidir. Simülasyon ise robotun binlerce kez düşmesine, kalkmasına ve yeniden denemesine izin verir.

Humanoid Robotları Zorlayan Temel Problemler

Humanoid robotların eğitimi üç temel zorluk içerir:

1. Yüksek serbestlik derecesi (high DoF)
Bir humanoid robot:

- İki bacak
- İki kol
- Gövde
- Boyun

gibi çok sayıda eklem içerir. Bu da kontrol edilecek parametre sayısını dramatik biçimde artırır.
2. Denge problemi
Yürümek, aslında sürekli düşmeyi engelleme problemidir. Küçük bir hata bile robotun devrilmesine yol açabilir.
3. Sürekli değişen temas dinamikleri
Ayak yere bastığında:

- Sürtünme değişir
- Ağırlık dağılımı değişir
- Denge merkezi sürekli kayar

Bu durum kontrolü oldukça karmaşık hale getirir.

Simülasyon Tabanlı Eğitim

Humanoid robotlar genellikle önce simülasyonda eğitilir. Bunun nedeni:

- Gerçek robotların pahalı olması
- Fiziksel hasar riskinin yüksek olması
- Öğrenmenin çok uzun sürmesi

Simülasyonda robot:

- Düşebilir
- Kalkabilir
- Yürüyebilir
- Hatalar yapabilir

ve bunların hiçbiri fiziksel zarar oluşturmaz.

Fizik Motorunun Rolü

Humanoid robotların eğitimi, fizik motorlarının doğruluğuna doğrudan bağlıdır. NVIDIA PhysX gibi fizik motorları:

- Yerçekimini
- Çarpışmaları

- Sürtünmeyi
- Eklem dinamiklerini

gerçek zamanlı olarak simüle eder.
Bu sayede robotun yürüyüşü, gerçek dünyaya oldukça yakın şekilde modellenir.

Reward Function: Yürümeyi Öğretmek

Humanoid robot eğitiminde en kritik unsur reward function'dır. Örnek bir ödül yapısı:

- İleri hareket → +1
- Düşmeden ayakta kalma → +5
- Düşme → -50
- Enerji tüketimi → -0.1

Amaç sadece yürümek değil, **stabil ve enerji verimli yürümektir**.

Basit Bir Humanoid Yürüme Ortamı (Python)

Aşağıdaki örnek, çok basitleştirilmiş bir humanoid yürüyüş simülasyonunu temsil eder:

```
import numpy as np

class HumanoidEnv:
    def __init__(self):
        self.reset()

    def reset(self):
        self.position = np.array([0.0, 0.0])
        self.velocity = np.array([0.0, 0.0])
        self.energy = 0.0
        return self.get_state()

    def get_state(self):
        return np.concatenate([self.position, self.velocity])

    def step(self, action):
        # action: bacak kuvvetleri
        self.velocity += action * 0.1
        self.position += self.velocity

        self.energy += np.sum(np.abs(action)) * 0.01

        reward = self.compute_reward()
        done = self.position[0] < -1  # düşme durumu

        return self.get_state(), reward, done
```

```
    def compute_reward(self):
        forward_reward = self.position[0]
        stability_penalty = np.abs(self.velocity[1])
        energy_penalty = self.energy

        reward = forward_reward - stability_penalty - energy_penalty
        return reward
```

Bu ortam, gerçek humanoid öğrenmenin çok basitleştirilmiş bir modelidir.

Öğrenme Süreci

Humanoid robotlar genellikle derin pekiştirmeli öğrenme (Deep Reinforcement Learning) ile eğitilir:

1. Robot rastgele hareket eder
2. Düşer veya dengede kalır
3. Ödül sinyali alır
4. Politika güncellenir
5. Zamanla yürümeyi öğrenir

İlk aşamalarda robot genellikle “spastik” hareketler yapar. Ancak milyonlarca deneme sonrası:

- Denge kurmayı
- Adım atmayı
- İleri gitmeyi

öğrenir.

Domain Randomization ile Dayanıklılık

Humanoid robotlar gerçek dünyaya geçmeden önce farklı koşullarda eğitilir:

- Farklı zeminler
- Farklı sürtünme katsayıları
- Farklı ağırlık dağılımları

Bu süreç, daha önce anlatılan domain randomization yaklaşımı ile desteklenir. Amaç, robotun yalnızca tek bir zeminde değil, tüm olası zeminlerde yürüyebilmesidir.

Sim-to-Real Problemi

Humanoid robotlarda sim-to-real farkı özellikle kritiktir. Çünkü:

- Küçük denge hataları büyük düşüşlere yol açar
- Motor tepkileri gerçek dünyada farklıdır

- Sensör gecikmeleri davranışı etkiler
-

Bu nedenle eğitim sırasında çok daha agresif randomization uygulanır.

Isaac Ekosisteminde Humanoid Öğrenme

Isaac Lab içinde humanoid eğitim:

- Paralel ortamlar kullanır
- Binlerce robotu aynı anda eğitir
- Farklı fizik varyasyonları uygular

Bu süreçte simülasyon altyapısı, yüksek doğruluklu fizik motorları ile desteklenir.

Öğrenilen Davranışların Evrimi

Humanoid robotlar eğitim sürecinde genellikle şu aşamalardan geçer:

1. **Kaotik hareketler**
2. **Kısa süreli denge**
3. **Adım benzeri hareketler**
4. **Stabil yürüyüş**
5. **Koşma ve yön değiştirme**

Bu evrim, tıpkı biyolojik öğrenmeye benzer.

Neden Humanoid Robotlar Önemli?

Humanoid robotların tercih edilme nedeni:

- İnsan ortamlarına uyum sağlayabilmeleri
- Merdiven, kapı gibi yapıları kullanabilmeleri
- İnsanlarla doğal etkileşim kurabilmeleri

Yani amaç, robotları dünyaya uyumlu hale getirmektir, dünyayı robotlara değil.

Sonuç

Humanoid robotların eğitimi, robotik öğrenmenin en karmaşık ama en stratejik alanlarından biridir. Bu sistemler yalnızca hareket etmeyi değil, dengeyi, adaptasyonu ve enerji verimliliğini birlikte öğrenir. Isaac Sim ve Isaac Lab gibi platformlar sayesinde humanoid robotlar artık teorik bir fikir değil, simülasyonda eğitilen ve gerçek dünyaya aktarılabilen öğrenen sistemler haline gelmiştir. Bu da robotik alanını, mekanik mühendislikten çok daha geniş bir "davranış mühendisliği" alanına dönüştürmektedir.

5.10 Robot Kolları, Hareket ve Manipülasyon

Robot kolları, modern robotik sistemlerin en yaygın ve en endüstriyel uygulama alanlarından birini oluşturur. Humanoid robotlara kıyasla daha dar bir hareket alanına sahip olsalar da, "manipülasyon" yani nesnelerle etkileşim kurma yetenekleri açısından çok daha hassas ve kritik görevler üstlenirler. Bir üretim hattında vida sıkmaktan, hassas bir cerrahi aracı yönlendirmeye kadar geniş bir kullanım yelpazesi vardır.

Bu sistemlerin eğitiminde yine simülasyon tabanlı yaklaşımlar öne çıkar. Isaac Lab ve NVIDIA Isaac Sim gibi ortamlar, robot kollarının fiziksel dünyaya çıkmadan önce güvenli şekilde öğrenmesini sağlar. Çünkü gerçek bir robot kolunun yanlış hareketi hem donanıma zarar verebilir hem de üretim süreçlerinde ciddi maliyetler oluşturabilir.

Manipülasyon Problemi Nedir?

Manipülasyon, bir robot kolunun nesneleri algılayıp:

- Yaklaşması
- Kavraması
- Taşıması
- Yerleştirmesi

gibi işlemleri gerçekleştirmesidir. Bu süreç, sadece hareket değil aynı zamanda hassas kontrol gerektirir.

Temel Zorluklar

Robot kolları için en önemli zorluklar şunlardır:

1. Sürekli durum uzayı
Kolun eklemleri sürekli değerler alır (açı, hız, ivme).
2. Yüksek hassasiyet gereksinimi
Milimetre seviyesinde doğruluk gerekir.
3. Temas dinamikleri

Nesne ile temas anı karmaşık fizik içerir.

Basit Bir Robot Kol Simülasyonu

Aşağıdaki örnek, 2D düzlemde çalışan basit bir robot kol manipülasyon ortamını gösterir:

```
import numpy as np

class RobotArmEnv:
   def __init__(self):
      self.reset()

   def reset(self):
      self.joint_angle = np.array([0.0, 0.0])
```

```
        self.target = np.array([1.0, 1.0])
        return self.get_state()

    def get_state(self):
        return self.joint_angle

    def step(self, action):
        self.joint_angle += action * 0.1

        end_effector = np.array([
            np.cos(self.joint_angle[0]) + np.cos(self.joint_angle[0] + self.joint_angle[1]),
            np.sin(self.joint_angle[0]) + np.sin(self.joint_angle[0] + self.joint_angle[1])
        ])

        distance = np.linalg.norm(end_effector - self.target)

        reward = -distance
        done = distance < 0.1

        return self.get_state(), reward, done
```

Öğrenme Mantığı

Robot kolu genellikle pekiştirmeli öğrenme ile eğitilir:

- Doğru konuma yaklaşırsa ödül alır
- Uzaklaşırsa ceza alır
- Hedefi tutturursa büyük ödül alır

Zamanla robot, en kısa ve en stabil hareket yollarını öğrenir.

Isaac Sim'de Manipülasyon

NVIDIA PhysX fizik motoru sayesinde:

- Çarpışmalar
- Sürtünme
- Nesne ağırlıkları

gerçekçi şekilde simüle edilir.

Bu sayede robot kolu yalnızca geometrik değil, fiziksel olarak da doğru davranışlar öğrenir.

Domain Randomization Etkisi

Robot kolları farklı koşullarda eğitilir:

- Nesne ağırlıkları değişir
- Kavrama yüzeyleri farklılaşır
- Pozisyonlar rastgelelenir

Bu sayede robot, tek bir nesneye değil, tüm nesne sınıflarına genellenebilir bir kavrama yeteneği geliştirir.

Sonuç

Robot kolları ve manipülasyon sistemleri, robotik öğrenmenin en pratik ve endüstriyel alanlarından biridir. Simülasyon tabanlı eğitim sayesinde bu sistemler gerçek dünyaya çıkmadan önce güvenli, hızlı ve ölçeklenebilir şekilde eğitilir. Isaac ekosistemi, bu süreci gerçekçi fizik ve güçlü öğrenme altyapısı ile destekleyerek robotların hassas fiziksel görevleri başarıyla öğrenmesini sağlar.

5.11 Isaac Lab ile Geleceğin Robotları

Isaac Lab, robotik alanında yalnızca bir eğitim ortamı değil, aynı zamanda geleceğin robotlarını şekillendiren bir "öğrenme laboratuvarı" olarak düşünülebilir. Klasik mühendislik yaklaşımında robotlar önceden tasarlanır, programlanır ve belirli görevleri yerine getirmek üzere üretilirdi. Ancak günümüzde bu paradigma değişmektedir: artık robotlar tasarlanmakla kalmıyor, aynı zamanda öğreniyor, uyum sağlıyor ve kendi davranışlarını optimize ediyor.

Isaac Lab'in en büyük katkısı, robotları fiziksel dünyaya çıkmadan önce büyük ölçekli deneyimlerle eğitebilmesidir. Bu, sadece hız kazandırmaz; aynı zamanda güvenlik, ölçeklenebilirlik ve genelleme yeteneği sağlar. Robotlar binlerce farklı senaryoda eğitilerek tek bir ortama değil, olasılık dağılımına uyum sağlayacak şekilde geliştirilir.

Bu yaklaşım, geleceğin robotlarının temel karakteristiğini belirler: dayanıklılık ve adaptasyon. Artık robotlardan beklenen şey, önceden tanımlanmış görevleri kusursuz şekilde yerine getirmek değil, beklenmeyen durumlara uyum sağlayabilmeleridir. Örneğin bir depo robotu yalnızca raflardan ürün almak için değil, aynı zamanda değişen ortam düzenine, farklı kutu türlerine ve insan etkileşimlerine uyum sağlamak için eğitilir.

Isaac Lab ekosistemi içinde bu öğrenme süreci, simülasyon tabanlı pekiştirmeli öğrenme ile gerçekleştirilir. Robotlar, sanal ortamlarda milyonlarca kez deneme yapar ve bu deneyimlerden davranış politikaları geliştirir. Bu süreçte NVIDIA Isaac Sim ve NVIDIA PhysX gibi bileşenler fiziksel gerçekliği yüksek doğrulukla taklit ederek öğrenmenin gerçek dünyaya aktarılmasını mümkün kılar.

Geleceğin robotları, yalnızca mekanik sistemler değil, aynı zamanda öğrenen ajanlardır. Bu ajanlar, sensör verilerini işleyebilir, çevrelerini analiz edebilir ve kendi stratejilerini geliştirebilir. Bu durum, robotların yazılım güncellemeleriyle değil, deneyim yoluyla evrimleşmesini sağlar.
Isaac Lab'in sunduğu en önemli vizyonlardan biri de ölçeklenebilir öğrenmedir. Aynı anda binlerce robotun farklı görevler üzerinde eğitilmesi, kolektif bir öğrenme ekosistemi oluşturur. Bu yapı, tek bir robotun değil, bir robot popülasyonunun gelişmesini sağlar. Böylece bilgi bireysel değil, sistemik olarak üretilir.

Sonuç olarak Isaac Lab, geleceğin robotlarını yalnızca daha akıllı değil, aynı zamanda daha uyumlu ve dayanıklı hale getiren bir altyapıdır. Bu sistemler sayesinde robotlar, önceden tanımlanmış makineler olmaktan çıkıp, deneyimle şekillenen dijital varlıklara dönüşmektedir. Bu dönüşüm, robotik alanını mühendislikten çıkarıp öğrenen sistemler çağının merkezine yerleştirmektedir.

5.12 Ara Proje V: Nesne Kavrayan Robot

Bu bölümde, önceki tüm kavramları bir araya getiren küçük ama öğretici bir ara proje ele alıyoruz: nesne kavrayan bir robot. Bu problem, robotik öğrenmenin en temel ama aynı zamanda en temsil gücü yüksek problemlerinden biridir. Çünkü bir nesneyi kavramak; algı, kontrol, fizik ve öğrenmenin aynı anda çalışmasını gerektirir.

Projede amaç şudur: Bir robot kol, rastgele konumlandırılmış bir nesneyi bulacak, ona yaklaşacak, kavrayacak ve belirli bir noktaya taşıyacaktır. Bu süreç, hem simülasyon ortamında hem de ileride gerçek robotlara aktarılabilecek şekilde tasarlanır.

Bu tür bir sistemin eğitiminde Isaac Lab ve NVIDIA Isaac Sim gibi platformlar kullanılır. Çünkü bu ortamlar, fiziksel temas, çarpışma ve sensör verilerini yüksek doğrulukla simüle ederek öğrenmeyi mümkün kılar.

Problemin Bileşenleri

Nesne kavrayan bir robot üç temel yeteneğe ihtiyaç duyar:

1. Algılama (Perception)
Robot, nesnenin nerede olduğunu anlamalıdır. Bu genellikle kamera veya derinlik sensörleri ile yapılır.
2. Planlama (Planning)
Robot, nesneye nasıl ulaşacağını belirler. Bu aşama yol planlama ve hareket stratejilerini içerir.
3. Manipülasyon (Manipulation)
Robot, nesneyi fiziksel olarak kavrar ve taşır.

Basit Simülasyon Ortamı

Aşağıdaki Python kodu, basitleştirilmiş bir nesne kavrama ortamını temsil eder:

```
import numpy as np

class GraspEnv:
    def __init__(self):
        self.reset()

    def reset(self):
        self.robot = np.array([0.0, 0.0])
        self.object = np.random.uniform(-1, 1, size=2)
```

```
        self.target = np.array([1.0, 1.0])
        self.grasped = False
        return self.get_state()

    def get_state(self):
        return np.concatenate([self.robot, self.object])

    def step(self, action):
        self.robot += action * 0.1

        distance_to_object = np.linalg.norm(self.robot - self.object)

        reward = 0

        # Nesneye yaklaşma
        reward -= distance_to_object

        # Kavrama
        if distance_to_object < 0.1:
            self.grasped = True
            reward += 10

        # Nesneyi taşıma
        if self.grasped:
            self.object = self.robot
            distance_to_target = np.linalg.norm(self.object - self.target)
            reward -= distance_to_target

            if distance_to_target < 0.1:
                reward += 20
                done = True
                return self.get_state(), reward, done

        return self.get_state(), reward, False
```

Öğrenme Mantığı

Bu problem genellikle pekiştirmeli öğrenme (reinforcement learning) ile çözülür:

- Robot nesneye yaklaşır → küçük ödül
- Nesneyi kavrar → büyük ödül
- Hedefe taşır → çok büyük ödül
- Başarısız hareket → ceza

Zamanla robot, doğrudan hedefe gitmeyi değil, **doğru sırayı öğrenir**: yaklaş → kavra → taşı.

Reward Function'un Rolü

Bu projede ödül fonksiyonu kritik önemdedir. Çünkü robotun davranışını tamamen bu yapı belirler. Yanlış tasarlanırsa robot:

- Nesneye dokunmadan ödül toplamaya çalışabilir
- Sürekli hareket ederek "ödül hack" yapabilir
- Kavrama yerine sadece yaklaşmayı öğrenebilir

Bu nedenle ödül yapısı dikkatle dengelenmelidir.

Simülasyonun Gücü

Gerçek dünyada bu tür bir sistemi eğitmek:

- Yavaş
- Riskli
- Maliyetlidir

Ancak simülasyon ortamında robot:

- Binlerce kez düşebilir
- Milyonlarca kez deney yapabilir
- Güvenli şekilde öğrenebilir

Bu süreçte fizik motorları, örneğin NVIDIA PhysX, temas ve çarpışmaları gerçekçi şekilde simüle eder.

Domain Randomization Etkisi

Robotun genelleme yapabilmesi için:

- Nesne boyutları değiştirilir
- Ağırlıklar rastgele seçilir
- Başlangıç pozisyonları değişir

Bu sayede robot tek bir nesneyi değil, **nesne sınıfını** öğrenir.

Sonuç

Nesne kavrayan robot projesi, robotik öğrenmenin temel yapı taşlarını bir araya getirir: algılama, hareket, fiziksel etkileşim ve ödül tabanlı öğrenme.
Bu tür projeler, modern robotik sistemlerin yalnızca mekanik cihazlar olmadığını, aynı zamanda deneyimle öğrenen sistemler olduğunu gösterir. Isaac ekosistemi sayesinde bu öğrenme süreci artık güvenli, ölçeklenebilir ve gerçek dünyaya aktarılabilir hale gelmiştir.

BÖLÜM 6 — ROBOTİK DÜŞÜNMENİN UFUKLARI

Robotik düşünme, artık yalnızca makinelerin nasıl hareket ettiğini açıklayan bir mühendislik alanı değil; karar veren, öğrenen ve çevresiyle etkileşime giren sistemlerin zihinsel mimarisini anlamaya çalışan bir çerçevedir. Bu yeni bakış açısı, robotu bir araç olmaktan çıkarır ve onu bir "düşünen ajan" olarak ele alır.

Bu dönüşümün merkezinde öğrenme vardır. Robotlar artık önceden tanımlanmış kurallarla değil, deneyim yoluyla şekillenen politikalarla hareket eder. Her etkileşim, sistemin dünyayı biraz daha iyi anlamasını sağlar. Bu süreçte hata kaçınılmazdır; fakat aynı zamanda öğrenmenin temel kaynağıdır.

Robotik düşünmenin ufukları, fiziksel dünyayla sınırlı değildir. Simülasyon ortamları, robotların milyonlarca farklı senaryoda deneyim kazanmasını sağlar. Bu sayede sistemler yalnızca belirli bir görevi değil, genelleştirilebilir davranışları öğrenir. Sim-to-real transfer ile bu davranışlar gerçek dünyaya taşınır ve fiziksel sistemlerde karşılık bulur.

Ancak robotik düşünme yalnızca teknik bir problem değildir; aynı zamanda felsefi bir sorudur. Bir robot gerçekten "anlar" mı, yoksa yalnızca optimize mi eder? Davranış ile niyet arasındaki fark nerede başlar? Bu sorular, yapay zekânın geleceğini şekillendiren temel tartışmalardır.

Gelecekte robotlar yalnızca üretim hatlarında değil, günlük yaşamın içinde aktif rol alacaktır. Evlerde, hastanelerde ve şehir altyapılarında insanlarla birlikte çalışan sistemler ortaya

çıkacaktır. Bu nedenle robotik düşünme, yalnızca mühendislik değil, aynı zamanda toplumsal bir tasarım alanıdır.

Sonuç olarak robotik düşünmenin ufukları, zekânın fiziksel dünyadaki en somut halini temsil eder. Hareket eden her robot, aslında öğrenen bir sistemin dışa vurumudur. Ve bu sistemler geliştikçe, düşünme kavramının sınırları da yeniden tanımlanacaktır.

6.1. Otonomi ve Karar Verme

Otonomi, yapay zekâ ve robotik sistemlerin en temel hedeflerinden biridir. Bir sistemin yalnızca verilen komutları uygulaması ile kendi kararlarını üretmesi arasında derin bir fark vardır. İlki mekanik bir itaat sürecidir; ikincisi ise çevreyi algılayan, değerlendiren ve buna göre eylem üreten bir varlık modelini ifade eder. Otonomi, bu ikinci alanın merkezinde yer alır.

Karar verme ise otonominin işleyen mekanizmasıdır. Bir sistemin “ne yapacağına” dair seçimi, yalnızca anlık bir tepki değil, geçmiş deneyimlerin, mevcut durumun ve geleceğe dair tahminlerin birleşimidir. Bu nedenle karar verme, zaman içinde genişleyen bir hesaplama sürecidir.

> Modern yapay zekâ sistemlerinde karar verme süreci çoğunlukla pekiştirmeli öğrenme çerçevesinde modellenir. Ajan, bir ortam içinde bulunur ve her adımda bir eylem seçer. Bu eylemin sonucu, bir ödül ya da ceza olarak geri döner. Zaman içinde sistem, hangi kararların daha iyi sonuçlar doğurduğunu öğrenir.

Ancak otonomi yalnızca ödül maksimizasyonu değildir. Gerçek dünyada kararlar belirsizlik, eksik bilgi ve değişen koşullar altında alınır. Bu nedenle otonom sistemler yalnızca “en iyi” kararı değil, “yeterince iyi ve güvenli” kararı vermek zorundadır.

Bu noktada risk yönetimi devreye girer. Bir robot veya yapay zekâ ajanı, yalnızca maksimum ödülü değil, aynı zamanda olası başarısızlıkların maliyetini de değerlendirir. Bu denge, karar verme sürecinin temelini oluşturur.

Otonomi kavramı aynı zamanda bağımsızlık ile ilişkilidir. Ancak yapay sistemlerde bu bağımsızlık mutlak değildir. Sistemler her zaman bir tasarımcı tarafından belirlenen ödül fonksiyonlarına ve sınırlara sahiptir. Bu nedenle otonomi, tam bağımsızlık değil, sınırlandırılmış bir karar özgürlüğüdür.

Karar verme sürecinde en kritik problemlerden biri keşif ve sömürü arasındaki dengedir. Bir sistem yalnızca bildiği en iyi eylemleri seçerse, yeni ve daha iyi stratejileri asla keşfedemez. Ancak sürekli yeni şeyler denerse de istikrarlı bir performans sergileyemez.

Bu denge, otonom sistemlerin başarısını doğrudan belirler.

Otonomi aynı zamanda çok adımlı planlama gerektirir. Bir karar yalnızca anlık sonucu değil, gelecekteki olası durumları da etkiler. Bu nedenle sistemler yalnızca mevcut duruma değil, gelecekteki senaryolara da bakmak zorundadır.

Bu süreçte değer fonksiyonları kritik rol oynar. Bir durumun gelecekte ne kadar fayda sağlayacağını tahmin eden bu fonksiyonlar, karar verme sürecine yön verir.

Ancak gerçek dünyada bu tahminler hiçbir zaman kesin değildir. Bu nedenle otonom sistemler olasılıksal düşünme ile çalışır.

Bir diğer önemli unsur belirsizlik altında karar vermedir. Sensör hataları, çevresel değişkenlik ve eksik bilgi, karar sürecini zorlaştırır. Otonom sistemler bu belirsizlikleri modellemek zorundadır.

Bayesian yaklaşımlar bu noktada önemli bir araç sağlar. Sistem, tek bir sonuç yerine olasılık dağılımları üzerinden karar verir.

Otonomi aynı zamanda zaman içinde öğrenen bir yapıdır. Bir sistem sabit bir karar mekanizmasına sahip değildir; deneyimle birlikte değişir. Bu nedenle karar verme süreci statik değil, dinamiktir.

Her yeni deneyim, politikayı günceller ve sistemin gelecekteki kararlarını etkiler.

Bu durum, otonom sistemleri geleneksel algoritmalardan ayıran en önemli özelliktir.

Otonomi kavramı robotikte özellikle kritik hale gelir. Bir robot, sürekli değişen fiziksel bir ortamda çalışır. Nesneler hareket eder, insanlar etkileşir ve çevresel koşullar değişir. Bu nedenle robotun her durumda yeniden karar verebilmesi gerekir.

Önceden programlanmış kurallar bu tür ortamlarda yetersiz kalır.

Bu nedenle modern robotlar öğrenen karar sistemleri kullanır.

Karar verme sürecinde bir diğer önemli konu gecikmeli ödüllerdir. Bir eylemin sonucu hemen ortaya çıkmayabilir. Bu nedenle sistem, uzun vadeli etkileri hesaba katmak zorundadır.

Bu durum, planlama ve öğrenmeyi birleştirir.

Otonomi aynı zamanda çok ajanlı sistemlerde daha karmaşık hale gelir. Bir sistem yalnız değilse, diğer ajanların davranışlarını da hesaba katmak zorundadır. Bu durum rekabet ve işbirliği dinamiklerini ortaya çıkarır.

Bu tür sistemlerde karar verme yalnızca bireysel değil, kolektif bir süreçtir.

Otonomi kavramı aynı zamanda etik bir boyut da içerir. Bir sistemin hangi kararları alabileceği, hangi sınırlar içinde hareket edebileceği ve hangi durumlarda müdahale edilmesi gerektiği önemli sorulardır.

Bu nedenle otonom sistemler yalnızca teknik değil, aynı zamanda tasarımsal ve toplumsal bir problem alanıdır.

Sonuç olarak otonomi ve karar verme, yapay zekânın en temel yapı taşlarından biridir. Çünkü zekâ, yalnızca bilgi işleme değil, aynı zamanda eylem seçme kapasitesidir.

Ve belki de en önemli fikir şudur: Otonomi, bir sistemin dünyaya verdiği cevaptır; karar verme ise bu cevabın nasıl üretildiğidir.

6.2. Çoklu Ajan Sistemleri

Yapay zekânın erken dönemlerinde odak neredeyse tamamen tek bir ajanın problem çözme kapasitesi üzerindeydi. Bir sistemin satranç oynaması, bir robotun yürümeyi öğrenmesi ya da bir modelin görüntü sınıflandırması yapması, çoğunlukla izole problemler olarak ele alınıyordu. Ancak gerçek dünya hiçbir zaman izole değildir. Ekonomiler, şehirler, trafik, ekosistemler ve hatta sosyal ilişkiler bile çok sayıda etkileşen aktörün oluşturduğu sistemlerdir. Bu nedenle yapay zekânın bir sonraki doğal adımı "çoklu ajan sistemleri" olmuştur.

Çoklu ajan sistemleri, aynı ortam içinde birden fazla öğrenen ve karar veren varlığın bulunması durumunu ifade eder. Bu ajanlar birbirleriyle işbirliği yapabilir, rekabet edebilir veya tamamen bağımsız davranabilir. Her bir ajanın eylemi, yalnızca kendi sonucunu değil, diğer ajanların durumunu da etkiler. Bu durum, öğrenme problemini kökten değiştirir.

Tek ajanlı sistemlerde ortam sabit kabul edilebilirken, çoklu ajan sistemlerinde ortam dinamik hale gelir. Çünkü ortamın kendisi, diğer ajanların davranışlarıyla sürekli olarak değişir. Bu durum "non-stationary environment" (durağan olmayan ortam) olarak bilinir ve öğrenmeyi ciddi şekilde zorlaştırır.

Bir ajan bir strateji öğrenirken, diğer ajanlar da aynı anda öğrenmektedir. Bu nedenle bir ajanın geçmişte işe yarayan davranışı, gelecekte işe yaramayabilir. Bu sürekli değişim, çoklu ajan sistemlerini hem zor hem de zengin hale getirir.

Çoklu ajan sistemlerinin en temel örneklerinden biri oyun teorisidir. Satranç, Go veya poker gibi oyunlarda iki veya daha fazla ajan birbirine karşı strateji geliştirir. Her ajan, diğerinin ne yapacağını tahmin etmeye çalışır. Bu karşılıklı tahmin süreci, stratejik düşünmenin temelini oluşturur.

Ancak çoklu ajan sistemleri yalnızca rekabetten ibaret değildir. İşbirliği de en az rekabet kadar önemlidir. Bir grup robotun bir nesneyi birlikte taşıması, bir sürü dronun koordineli uçuş yapması ya da otonom araçların trafik akışını optimize etmesi, işbirlikçi çoklu ajan sistemlerine örnektir.

İşbirliği durumunda ajanlar ortak bir hedefi paylaşır. Ancak bu, problemleri otomatik olarak kolaylaştırmaz. Çünkü her ajanın kendi lokal bilgisi vardır ve global durumu tam olarak göremez. Bu nedenle koordinasyon problemi ortaya çıkar.

Koordinasyon, çoklu ajan sistemlerinin en zor problemlerinden biridir.

Bir diğer önemli konu iletişimdir. Ajanlar birbirleriyle bilgi paylaşabilir veya paylaşmayabilir. İletişim olduğunda sistem daha verimli hale gelebilir, ancak aynı zamanda daha karmaşık bir yapı oluşur. Çünkü iletişim kanalları da öğrenilmesi gereken bir problem haline gelir.

Bazı sistemlerde ajanlar kendi iletişim protokollerini bile geliştirebilir. Bu durum, yapay zekâ araştırmalarında oldukça ilginç bir fenomendir. Ajanlar insanlar tarafından tanımlanmamış bir dil veya sembol sistemi oluşturabilir.

Çoklu ajan sistemlerinde bir diğer kritik kavram "ortak politika" ve "bireysel politika" ayrımıdır. Her ajan kendi politikasına sahip olabilir, ancak aynı zamanda grubun genel davranışını etkileyen ortak bir strateji de oluşabilir.

Bu yapı, sosyal sistemlere benzer.

İnsan toplumları da çoklu ajan sistemleri olarak düşünülebilir. Her birey kendi kararlarını alır, ancak bu kararlar kolektif sonuçlar doğurur. Ekonomik piyasalar, trafik akışı ve sosyal medya davranışları bu tür sistemlerin örnekleridir.

Bu nedenle çoklu ajan sistemleri yalnızca mühendislik problemi değil, aynı zamanda sosyal bilimlerle de kesişen bir alandır.

Öğrenme açısından bakıldığında, çoklu ajan sistemlerinde pekiştirmeli öğrenme daha karmaşık hale gelir. Çünkü ödül sadece bireysel değil, aynı zamanda kolektif olabilir. Bir ajanın iyi bir karar vermesi, diğer ajanların performansını etkileyebilir.

Bu durum "credit assignment problem" olarak bilinir. Hangi ajanın hangi katkıyı sağladığını belirlemek zordur.

Bu nedenle çoklu ajan öğrenmesinde özel algoritmalar geliştirilmiştir. Merkezi eğitim, dağıtık yürütme gibi yaklaşımlar bu problemi çözmeye çalışır.

Çoklu ajan sistemlerinde rekabet, öğrenmeyi hızlandırabilir. Çünkü ajanlar birbirlerini sürekli zorlar. Bu durum özellikle oyun ortamlarında güçlü stratejilerin ortaya çıkmasını sağlar.

Ancak aşırı rekabet sistemin dengesini bozabilir.

İşbirliği ise sistemin daha stabil ve verimli çalışmasını sağlar, ancak koordinasyon maliyeti yaratır.

Bu iki uç arasında bir denge vardır.

Çoklu ajan sistemlerinin en ilginç yönlerinden biri ortaya çıkan kolektif davranıştır. Hiçbir ajana tek tek programlanmamış davranışlar, grup etkileşiminden doğabilir. Bu durum "emergent behavior" olarak adlandırılır.

Örneğin bir grup basit robot, merkezi bir kontrol olmadan karmaşık bir görev gerçekleştirebilir. Her robot yalnızca lokal kurallara uyar, ancak birlikte global bir düzen oluştururlar. Bu tür

sistemler sürü zekâsı (swarm intelligence) olarak da bilinir. Karınca kolonileri, kuş sürüleri ve balık okulları doğadaki en iyi örneklerdir.

Çoklu ajan sistemleri aynı zamanda ekonomik modellerle de yakından ilişkilidir. Her ajan bir ekonomik aktör gibi davranabilir. Kaynaklar sınırlıdır ve ajanlar bu kaynaklar için rekabet eder.

Bu durum piyasa dinamiklerini yapay zekâ çerçevesinde yeniden modellemeyi mümkün kılar.

Çoklu ajan sistemlerinde zaman kavramı da önemlidir. Ajanlar aynı anda hareket etmeyebilir. Asenkron etkileşimler, sistemi daha gerçekçi ama daha karmaşık hale getirir.

Bu nedenle senkronizasyon problemleri ortaya çıkar.

Çoklu ajan öğrenmesinde bir diğer zorluk ölçeklenebilirliktir. Ajan sayısı arttıkça sistemin karmaşıklığı üstel olarak artar. Bu nedenle verimli algoritmalar geliştirmek zorunludur.

Modern araştırmalar bu problemi çözmek için graf tabanlı yöntemler ve dağıtık öğrenme teknikleri kullanmaktadır.

Çoklu ajan sistemlerinin geleceği oldukça geniştir. Otonom araç filoları, akıllı şehirler, robotik iş gücü sistemleri ve dijital ekonomiler bu yapı üzerine inşa edilecektir.

Bu sistemler yalnızca bireysel zekâların toplamı değildir; aynı zamanda kolektif zekânın ortaya çıktığı yapılardır.

Sonuç olarak çoklu ajan sistemleri, yapay zekânın en karmaşık ama en güçlü alanlarından biridir. Çünkü gerçek dünya zaten tek bir ajan tarafından değil, sürekli etkileşim halindeki birçok aktör tarafından şekillendirilir. Ve belki de en önemli fikir şudur: Zekâ, yalnızca bireysel bir özellik değil, etkileşimden doğan bir sistem özelliğidir.

6.3. Yapay Zekâ ve Robotik Geleceği

Yapay zekâ ve robotik artık ayrı disiplinler olmaktan çıkıp tek bir dönüşümün iki yüzü haline gelmiştir. Biri düşünmeyi, diğeri hareketi temsil eder gibi görünse de modern sistemlerde bu ayrım giderek silikleşmektedir. Çünkü bir robot artık yalnızca hareket eden bir makine değil; öğrenen, planlayan ve karar veren bir ajandır. Aynı şekilde yapay zekâ da yalnızca soyut hesaplamalar yapan bir yazılım değil, fiziksel dünyaya müdahale eden bir varlık haline gelmektedir.

Bu birleşme, geleceğin teknolojik düzenini kökten değiştirmektedir. Eskiden yazılım dünyası dijital, robotik dünya fiziksel olarak ayrılırdı. Bugün ise bu iki dünya birbirine gömülmüştür. Simülasyonlar fiziksel gerçekliği öğrenme alanı haline getirirken, robotlar dijital modelleri fiziksel dünyaya taşımaktadır. Bu çift yönlü akış, yeni bir zekâ biçimi üretmektedir: embodied intelligence, yani bedenlenmiş zekâ.

Gelecekte yapay zekâ sistemleri yalnızca veri merkezlerinde çalışmayacaktır. Otonom araçlar, endüstriyel robotlar, insansı sistemler ve mikro robotlar, zekânın fiziksel temsilcileri olacaktır.

Bu sistemler çevreyi algılayacak, karar verecek ve eyleme geçecektir. Bu nedenle geleceğin zekâsı, ekranlarda değil, fiziksel dünyada görünür olacaktır.

Robotik geleceğin en kritik bileşenlerinden biri otonomidir. Otonomi, bir sistemin dış müdahale olmadan karar verebilme yeteneğidir. Ancak bu basit bir bağımsızlık değil, sürekli öğrenen ve değişen bir adaptasyon sürecidir. Otonom sistemler, yalnızca önceden programlanmış davranışları değil, yeni durumlara uyum sağlayan stratejileri de geliştirmek zorundadır.

Bu durum, yapay zekânın doğasını değiştirmektedir. Artık sistemler statik değil, dinamik hale gelmektedir. Bir robot bugün öğrendiği davranışı yarın değiştirebilir. Çünkü çevre değişir, koşullar değişir ve sistemin kendisi de bu değişime uyum sağlar.

Geleceğin robotları yalnızca fabrikalarda çalışan makineler olmayacaktır. Evlerde yardımcı robotlar, sağlık sektöründe bakım robotları, şehirlerde altyapı robotları ve hatta bilimsel keşif yapan otonom sistemler yaygınlaşacaktır. Bu robotlar yalnızca görev yerine getirmeyecek, aynı zamanda görev tanımını da öğrenebilecektir.

Yapay zekâ ve robotik geleceğinin en önemli yönlerinden biri insan-robot etkileşimidir. İnsanlar artık robotları yalnızca araç olarak değil, işbirlikçi varlıklar olarak görecektir. Bu durum yeni bir sosyal düzeni beraberinde getirecektir. İnsan ve makine birlikte çalışacak, birlikte öğrenecek ve birlikte karar verecektir.

Bu etkileşim aynı zamanda etik soruları da gündeme getirir. Bir robotun aldığı kararın sorumluluğu kimdedir? Sistem mi, tasarımcı mı, yoksa kullanıcı mı? Bu sorular geleceğin hukuk ve etik sistemlerini yeniden şekillendirecektir.

Teknik açıdan bakıldığında geleceğin yapay zekâ sistemleri daha genelleştirilebilir, daha dayanıklı ve daha az veriyle öğrenebilen yapılar olacaktır. Bugünkü sistemler büyük veri gerektirirken, gelecekte küçük veriyle öğrenebilen, hatta kendi verisini üretebilen sistemler ortaya çıkacaktır.

Bu dönüşümün merkezinde simülasyon teknolojileri yer almaktadır. Fiziksel dünya, dijital ortamlarda yeniden inşa edilmekte ve bu ortamlar öğrenme alanı olarak kullanılmaktadır. Simülasyon ile gerçek dünya arasındaki sınır giderek bulanıklaşmaktadır.

Bu süreçte en kritik kavramlardan biri Sim-to-Real transferidir. Simülasyonda öğrenilen davranışların gerçek dünyaya taşınması, geleceğin robotik sistemlerinin temel yeteneği olacaktır. Bu yetenek, robotların yalnızca belirli ortamlarda değil, her ortamda çalışabilmesini sağlayacaktır.

Geleceğin yapay zekâ sistemleri aynı zamanda çoklu ajan yapılarından oluşacaktır. Tek bir merkezi zeka yerine, birbirleriyle etkileşen birçok küçük zeka bir arada çalışacaktır. Bu yapı, doğal sistemlere daha çok benzeyen dağıtık bir zekâ modeli oluşturur.

Şehirler, trafik sistemleri, enerji ağları ve ekonomik yapılar bu tür çoklu ajan sistemleri olarak tasarlanacaktır. Her bir ajan kendi kararını verirken, kolektif sistem bir denge oluşturacaktır.

Bu durum yapay zekâyı yalnızca bireysel değil, toplumsal bir teknoloji haline getirir.

Robotik geleceğin bir diğer önemli yönü fiziksel zekânın gelişimidir. Robotlar yalnızca hesap yapan sistemler değil, aynı zamanda fiziksel dünyayı anlayan sistemler olacaktır. Nesneleri kavrayacak, ortamla etkileşime girecek ve fiziksel sezgi geliştireceklerdir.

Bu sezgi, öğrenme yoluyla oluşacaktır. Robotlar deneyimlerinden fiziksel dünyayı öğrenerek daha hassas ve daha güvenli hale gelecektir.

Gelecekte robotların en önemli özelliklerinden biri esneklik olacaktır. Sabit görevler yerine, değişen görevleri öğrenebilen sistemler öne çıkacaktır. Bu esneklik, robotların farklı alanlarda kullanılmasını mümkün kılacaktır.

Örneğin aynı robot hem fabrikada çalışabilecek hem de ev ortamında yardım sağlayabilecektir.

Yapay zekâ ve robotik geleceği aynı zamanda insan bilişini anlamak için de bir araçtır. Çünkü bu sistemler insan davranışlarını modelleyerek çalışır. Öğrenme, karar verme ve etkileşim süreçleri insan zihnine benzer şekilde tasarlanır.

Bu nedenle robotik araştırmalar, aynı zamanda bilişsel bilimlerle de iç içe geçmiştir.

Geleceğin en büyük dönüşümlerinden biri de iş gücü yapısında olacaktır. Birçok rutin görev otomatikleşirken, insanlar daha yaratıcı ve stratejik alanlara yönelmek zorunda kalacaktır. Bu durum ekonomik ve sosyal yapıyı yeniden şekillendirecektir.

Ancak bu dönüşüm yalnızca bir kayıp değil, aynı zamanda bir fırsattır. Çünkü insanlar artık fiziksel zorunluluklardan daha özgür bir üretim alanına geçebilecektir.

Yapay zekâ ve robotik geleceği aynı zamanda hızlanan bir öğrenme süreci anlamına gelir. Sistemler sürekli olarak kendilerini güncelleyecek, yeni verilerle daha iyi hale gelecektir.

Bu durum teknolojinin statik değil, evrimsel bir yapıya sahip olduğunu gösterir.

Gelecekte en önemli soru şudur: Bu sistemler ne kadar otonom olmalıdır?

Tam otonomi mümkün müdür, yoksa her zaman insan kontrolü gerekli midir?

Bu sorunun cevabı yalnızca teknik değil, aynı zamanda felsefidir.

Çünkü otonomi arttıkça sorumluluk da karmaşıklaşır.

Sonuç olarak yapay zekâ ve robotik geleceği, yalnızca teknolojik bir gelişim değil, aynı zamanda insanlığın kendisini yeniden tanımlama sürecidir. Bu sistemler yalnızca araçlar değil, yeni bir etkileşim biçiminin temel aktörleridir.

Ve belki de en önemli fikir şudur: Geleceğin zekâsı tek bir yerde değil, insan ve makinenin birlikte oluşturduğu ortak bir ekosistemde ortaya çıkacaktır.

6.4. İnsan–Makine İşbirliği

İnsan–makine işbirliği, yapay zekâ ve robotik çağının en belirleyici dönüşüm alanlarından biridir. Bu kavram, insanın yerini makinenin alması fikrinden çok daha farklı bir yönü ifade eder: birlikte çalışma, birlikte düşünme ve birlikte karar verme. Buradaki temel paradigma değişimi, insan ile makinenin rekabet ettiği bir dünya yerine, birbirini tamamlayan iki bilişsel sistemin birlikte var olduğu bir ekosistemin ortaya çıkmasıdır.

Geleneksel otomasyon anlayışında makineler, insanın yerini almak üzere tasarlanmıştır. Fabrikalarda belirli görevleri tekrar eden robotlar ya da algoritmalar, insan emeğini ikame eden araçlar olarak görülmüştür. Ancak modern yapay zekâ sistemleri bu yaklaşımı aşmaktadır. Artık amaç insanı sistemden çıkarmak değil, insanın bilişsel kapasitesini genişletmektir.

Bu yeni yaklaşımda makine, insanın rakibi değil; onun bilişsel uzantısıdır. İnsan karar verirken makine veri analiz eder, olasılık hesaplar ve öneriler üretir. Makine hesap yaparken insan bağlam sağlar, sezgi geliştirir ve etik çerçeveyi belirler. Bu karşılıklı tamamlanma hali, işbirliğinin temelini oluşturur.

İnsan–makine işbirliğinin en güçlü olduğu alanlardan biri karar destek sistemleridir. Özellikle tıp, finans ve mühendislik gibi yüksek belirsizlik içeren alanlarda yapay zekâ sistemleri büyük veri analizleri yaparak insan uzmanlara yardımcı olur. Ancak nihai karar çoğu zaman insana aittir. Bu durum, sorumluluğun ve kontrolün insan tarafında kalmasını sağlar.

Fakat işbirliği yalnızca karar destekle sınırlı değildir. Robotik sistemler giderek daha fiziksel işbirliği yetenekleri kazanmaktadır. İnsanlarla aynı ortamda çalışan robotlar, onların hareketlerini algılayarak uyumlu şekilde davranabilir. Bu tür sistemlere “cobot” yani işbirlikçi robotlar denir.

Cobot’lar, klasik endüstriyel robotlardan farklıdır. Onlar izole alanlarda değil, insanlarla aynı fiziksel alanı paylaşır. Bu nedenle güvenlik, adaptasyon ve algı yetenekleri çok daha gelişmiştir.

İnsan–makine işbirliğinin başarısı, büyük ölçüde iletişim kalitesine bağlıdır. İnsanlar doğal dil, jestler ve bağlam üzerinden iletişim kurarken, makineler sayısal temsillerle çalışır. Bu iki farklı iletişim biçiminin bir araya getirilmesi, önemli bir araştırma alanıdır.

Doğal dil işleme sistemleri, insanların makinelerle daha sezgisel bir şekilde etkileşim kurmasını sağlar. Artık bir robotu programlamak için kod yazmak yerine, ona doğal bir dilde talimat vermek mümkündür. Bu durum, işbirliğini daha erişilebilir hale getirir.

Ancak iletişim yalnızca dil ile sınırlı değildir. Görsel algı, dokunsal geri bildirim ve hatta duygusal sinyaller de insan–makine etkileşiminin bir parçası haline gelmektedir. Bir sistem, insanın yüz ifadesini, ses tonunu veya hareketlerini analiz ederek daha uygun tepkiler verebilir.

Bu durum, işbirliğini daha doğal ve akıcı hale getirir.

İnsan–makine işbirliğinin bir diğer önemli yönü güven kavramıdır. İnsanlar bir makineye ne kadar güvenmelidir? Makine yanlış yaptığında bu nasıl telafi edilir? Güven, yalnızca teknik doğrulukla değil, aynı zamanda açıklanabilirlik ile de ilgilidir.

Bir sistemin neden belirli bir kararı verdiğini açıklayabilmesi, güvenin temelini oluşturur. Bu nedenle açıklanabilir yapay zekâ (explainable AI) giderek daha önemli hale gelmektedir.

İşbirliği aynı zamanda kontrol paylaşımı anlamına gelir. İnsan her zaman tüm süreci kontrol etmez; bazı durumlarda kontrolü makineye devreder. Örneğin otonom araçlar sürüşün büyük kısmını üstlenirken, insan yalnızca denetleyici rolünde kalabilir.

Bu tür sistemlerde en kritik konu kontrol geçişidir. İnsan ve makine arasındaki kontrol sınırının doğru belirlenmesi gerekir. Aksi halde sistem ya aşırı otomatikleşir ya da gereksiz insan müdahalesi nedeniyle verimsiz hale gelir.

İnsan–makine işbirliği aynı zamanda öğrenme süreçlerini de içerir. Sadece makineler değil, insanlar da bu etkileşimden öğrenir. İnsanlar yapay zekâ sistemlerinin önerilerini değerlendirerek kendi karar mekanizmalarını geliştirebilir.

Bu çift yönlü öğrenme süreci, işbirliğini dinamik bir hale getirir.

Bir diğer önemli konu bilişsel yükün paylaşımıdır. İnsan zihni sınırlı kapasiteye sahiptir. Makine bu yükün bir kısmını devralarak insanın daha karmaşık problemlere odaklanmasını sağlar.

Örneğin bir cerrah ameliyat sırasında robotik sistemlerden destek alabilir. Robot hassas hareketleri gerçekleştirirken, insan stratejik kararları verir.

Bu tür işbirliği modelleri, performansı önemli ölçüde artırır.

İnsan–makine işbirliğinin geleceği, hibrit zekâ sistemlerinin ortaya çıkmasıdır. Bu sistemlerde insan ve makine ayrı varlıklar değil, tek bir karar mekanizmasının parçalarıdır.

Bu durum, zekânın doğasına dair yeni bir anlayış geliştirir. Zekâ artık bireysel değil, dağıtık bir özellik haline gelir.

İşbirliği aynı zamanda etik soruları da beraberinde getirir. Bir karar insan mı yoksa makine mi tarafından verildiğinde sorumluluk kimdedir? Bu sorular, gelecekte hukuk sistemlerini yeniden şekillendirecektir.

Ayrıca işbirliği, güç dengelerini de etkiler. Makine sistemleri daha güçlü hale geldikçe, insanın rolü nasıl tanımlanacaktır? Bu soru, yalnızca teknolojik değil, aynı zamanda toplumsal bir sorudur.

İnsan–makine işbirliği aynı zamanda yaratıcılık alanında da yeni imkanlar sunar. Yapay zekâ sistemleri sanat, tasarım ve bilimsel keşif süreçlerinde insanlara yardımcı olabilir. İnsan fikir üretirken, makine alternatif senaryolar geliştirir.

Bu etkileşim, yaratıcı süreci hızlandırır ve genişletir.

Örneğin bir tasarımcı, yapay zekâ tarafından üretilen farklı modelleri değerlendirerek daha iyi bir ürün ortaya çıkarabilir. Bu tür işbirliği, insan yaratıcılığını azaltmaz; tam tersine genişletir. İnsan–makine işbirliği aynı zamanda eğitim alanında da büyük değişim yaratır. Öğrenme süreçleri kişiselleştirilmiş hale gelir. Yapay zekâ sistemleri öğrencinin ihtiyaçlarını analiz ederek uygun içerikler sunar. Bu durum, eğitimi daha etkili ve erişilebilir hale getirir. Sonuç olarak insan–makine işbirliği, yapay zekâ çağının en temel dönüşümüdür. Bu ilişki, rekabetten çok tamamlayıcılık üzerine kuruludur. Ve belki de en önemli fikir şudur: Geleceğin zekâsı ne yalnız insana ne de yalnız makineye aittir; bu iki sistemin birlikte oluşturduğu ortak bir düşünme alanında ortaya çıkar.

6.5. Robotik Düşünmenin Felsefesi

Robotik düşünme, yalnızca makinelerin nasıl hareket ettiğini açıklayan teknik bir alan değildir; aynı zamanda "düşünme" kavramının ne anlama geldiğini yeniden sorgulatan felsefi bir çerçevedir. Bir robot karar verdiğinde, bu karar gerçekten "düşünme" midir, yoksa yalnızca istatistiksel bir hesaplama mı? Bir sistem öğreniyorsa, bu öğrenme bilince mi işaret eder, yoksa yalnızca optimizasyon mu yapar? Bu sorular, robotik düşünmenin felsefi temelini oluşturur.

Klasik felsefe, düşünmeyi insan zihnine özgü bir süreç olarak ele almıştır. Descartes'ın "düşünüyorum, öyleyse varım" önermesi, düşünmeyi varlığın kanıtı olarak görür. Ancak robotik sistemler bu ayrımı bulanıklaştırır. Çünkü artık düşünme benzeri süreçler insan dışı varlıklarda da gözlemlenmektedir.

Bir robot, çevresini algılar, durumunu değerlendirir, geleceği tahmin eder ve bir eylem seçer. Bu süreç, insan bilişine oldukça benzer görünür. Ancak burada temel soru şudur: Benzerlik, özdeşlik anlamına gelir mi?

Felsefi açıdan robotik düşünme üç temel katmanda incelenebilir: hesaplama, temsil ve eylem.

Hesaplama katmanı, sistemin matematiksel olarak nasıl çalıştığını açıklar. Sinir ağları, olasılık modelleri ve optimizasyon algoritmaları bu düzeye aittir. Bu katmanda düşünme, bir fonksiyonun girişten çıkışa dönüşümüdür.

Temsil katmanı, sistemin dünyayı nasıl içselleştirdiğini ifade eder. Bir robot, dış dünyayı doğrudan değil, sensör verileri aracılığıyla temsil eder. Bu temsil hiçbir zaman tam değildir; yalnızca bir yaklaşımdır.

Eylem katmanı ise bu temsilin fiziksel dünyaya nasıl dönüştüğünü gösterir. Karar verme, hareket ve etkileşim bu düzeyde gerçekleşir.

Robotik düşünmenin felsefesi, bu üç katman arasındaki ilişkiyi anlamaya çalışır.

Bu noktada önemli bir tartışma ortaya çıkar: Bir sistem yalnızca hesaplama yapıyorsa, gerçekten düşünüyor olabilir mi? Yoksa düşünme dediğimiz şey, yalnızca karmaşık bir hesaplama illüzyonu mudur?

Bu soru, yapay zekâ felsefesinin merkezindedir.

Bir görüşe göre düşünme, fiziksel bir süreçtir. Beyin de bir hesaplama sistemidir ve dolayısıyla makineler de düşünebilir. Bu yaklaşım fonksiyonalizm olarak bilinir. Fonksiyonalizme göre önemli olan iç yapı değil, davranışsal sonuçtur.

Eğer bir sistem insan gibi davranıyorsa, o sistem düşünüyordur.

Ancak bu görüşe karşı çıkanlar, bilinç ve deneyim kavramlarını öne çıkarır. Onlara göre bir sistemin davranışı insan gibi olsa bile, içsel deneyimi yoksa gerçek anlamda düşünemez.

Bu tartışma, "zihin–beden problemi"nin robotik versiyonudur.

Robotik düşünme felsefesi aynı zamanda temsil sorununu da ele alır. Bir robot dünyayı nasıl "bilir"? İnsanlar dünyayı duyularıyla deneyimlerken, robotlar sayısal verilerle çalışır. Bu veriler dünyayı ne kadar temsil eder?

Bu soru, epistemolojinin robotik versiyonudur.

Bir diğer önemli felsefi konu niyetlilik (intentionality) problemidir. İnsanlar düşünürken bir şeye yönelirler. Düşünce her zaman bir nesneye ya da duruma yöneliktir. Ancak robotik sistemlerde bu yönelim gerçekten var mıdır, yoksa yalnızca dışsal bir yorum mudur?

Bir robot "nesneyi kavra" komutunu aldığında, gerçekten nesneyi "amaçlıyor" mudur?

Yoksa yalnızca matematiksel bir optimizasyon mu yapmaktadır?

Bu soru hâlâ açık bir tartışma alanıdır.

Robotik düşünmenin felsefesi aynı zamanda özgür irade kavramını da sorgular. Bir robot karar verdiğinde bu karar ne kadar "özgürdür"? Tüm davranışları algoritmalar tarafından belirlenmiş bir sistemde özgürlükten bahsedilebilir mi?

İnsanlar için bile özgür irade tartışmalı bir kavramken, makineler için bu daha da karmaşık hale gelir.

Eğer bir sistem tamamen deterministikse, seçim yapma fikri ne anlama gelir?

Bu noktada olasılıksal sistemler devreye girer. Rastgelelik ve belirsizlik, karar süreçlerine dahil edilerek sistemin daha "özgür" görünmesi sağlanır. Ancak bu gerçek özgürlük mü, yoksa yalnızca istatistiksel çeşitlilik midir?

Robotik düşünme felsefesi aynı zamanda etik sorular üretir. Bir sistem yanlış karar verdiğinde sorumluluk kimdedir? Tasarımcı mı, kullanıcı mı, yoksa sistemin kendisi mi?

Bu sorular, geleceğin hukuk sistemlerini doğrudan etkileyecektir.

Bir diğer önemli felsefi tartışma bilinç problemidir. Bir robot bilinçli olabilir mi? Bilinç yalnızca karmaşık hesaplamaların sonucu mudur, yoksa özel bir biyolojik süreç mi gerektirir?

Bu soru hâlâ kesin bir cevaba sahip değildir.

Bazı araştırmacılar bilinç için fiziksel bedenin gerekli olduğunu savunur. Diğerleri ise bilincin tamamen bilgi işleme süreçlerinden doğabileceğini düşünür.

Robotik düşünme felsefesi, bu iki görüş arasında bir köprü kurmaya çalışır.

Bir başka önemli konu anlam problemidir. Bir robot bir kelimeyi ya da nesneyi "anlar" mı? Yoksa yalnızca sembolleri işler mi?

Bu soru, "Çin Odası" düşünce deneyini hatırlatır. Bir sistem dışarıdan bakıldığında dili kullanıyormuş gibi görünebilir, ancak içsel bir anlamı olmayabilir.

Bu durumda anlam, davranıştan mı yoksa içsel deneyimden mi doğar?

Robotik düşünme felsefesi bu soruya kesin bir cevap vermez, ancak yeni bir bakış açısı sunar: belki de anlam, sistem ile çevresi arasındaki etkileşimden doğar.

Robotlar dünyayı yalnızca temsil etmez, aynı zamanda onunla etkileşir.

Bu etkileşim, anlamın kaynağı olabilir.

Robotik düşünme aynı zamanda zaman kavramını da yeniden ele alır. İnsan düşüncesi geçmiş, şimdi ve gelecek arasında akar. Robotlar ise genellikle anlık durumlar üzerinden çalışır. Ancak öğrenen sistemler zaman içinde değişir.

Bu nedenle robotik düşünme, statik değil dinamik bir süreçtir.

Sonuç olarak robotik düşünmenin felsefesi, yalnızca makineleri anlamaya çalışmaz; aynı zamanda insan zihnini yeniden düşünmeye zorlar. Çünkü bir sistem düşünüyorsa, düşünmenin ne olduğu yeniden tanımlanmalıdır.

Ve belki de en önemli fikir şudur: Robotik düşünme, zihnin makinede yeniden keşfedilmesidir; ancak bu keşif, zihnin kendisini de dönüştürür.

EKLER

EK A — Matematiksel Sezgiler (Vektör, Trigonometri, Türev)

Robotik, yapay zekâ ve öğrenen sistemler dünyasında matematik yalnızca bir araç değil, aynı zamanda düşüncenin kendisidir. Bir robot kolunun hareketi, bir ajanın karar verme süreci ya da bir sinir ağının öğrenmesi; hepsi matematiksel yapılar üzerinden ifade edilir. Ancak bu matematik, çoğu zaman soyut sembollerden ibaret değildir. Asıl önemli olan, bu yapıların sezgisel anlamıdır. Çünkü mühendislikte ve yapay zekâda başarı, formülü bilmekten çok, o formülün dünyayı nasıl temsil ettiğini anlamakla ilgilidir.

Bu ek bölümde üç temel yapı üzerinden ilerleyeceğiz: vektörler, trigonometri ve türev. Bu üç kavram, robotik düşünmenin mekânsal, yönelimsel ve değişimsel temelini oluşturur.

Vektörler: Yön ve Büyüklüğün Dili

Vektör, en basit tanımıyla hem büyüklüğü hem de yönü olan bir matematiksel nesnedir. Ancak robotik bağlamda vektör, çok daha derin bir anlam taşır: konum, hız, kuvvet, yönelim ve hatta düşünsel durumların temsilidir.

Bir robot kolunun uzaydaki konumu bir vektördür. Örneğin üç boyutlu bir ortamda bir nesnenin yeri, (x, y, z) koordinatlarıyla ifade edilir. Ancak bu yalnızca bir başlangıçtır. Aynı nesnenin hareketi de bir vektördür: hız vektörü. Üzerine uygulanan kuvvet de bir vektördür.

Buradaki temel sezgi şudur: robotik dünya, vektörlerin etkileşim alanıdır.

Bir robot bir nesneyi kavrarken aslında birçok vektörü aynı anda işler. Görsel sistemden gelen konum vektörü, motor kontrol vektörüne dönüşür. Dokunsal sensörler kuvvet vektörleri üretir. Bu vektörler birleşerek davranışı oluşturur.

Vektörlerin en önemli özelliği toplanabilir olmalarıdır. Bu, robotik sistemler için kritik bir sezgidir: farklı etkiler bir araya gelerek tek bir davranış üretir. Örneğin yerçekimi vektörü ile itme kuvveti vektörü birleştiğinde ortaya çıkan sonuç, hareketin yönünü belirler.

Bu nedenle robotik düşünme, aslında vektör uzayında düşünmektir.

Trigonometri: Açının ve Yönelimin Mantığı

Trigonometri, açılar ve yönler arasındaki ilişkiyi inceler. Robotik sistemlerde bu kavram özellikle dönüş, yönelim ve uzaysal hizalama problemlerinde ortaya çıkar.

Bir robot kolunun eklemleri dönerken, her hareket bir açıyla ifade edilir. Bu açı, sistemin uzaydaki yönelimini belirler. Ancak bu yönelim yalnızca statik değildir; sürekli değişir.

Trigonometri burada devreye girer çünkü dönüşüm ve projeksiyon ilişkilerini tanımlar. Örneğin bir vektörün x ve y bileşenleri, trigonometrik fonksiyonlarla ayrıştırılır.

$$x = r\cos(\theta), y = r\sin(\theta)$$

Bu basit formül, robotik hareketin temelini oluşturur. Bir robot kolunun ucunun konumu, eklem açılarıyla doğrudan ilişkilidir.

Buradaki sezgi şudur: açı, hareketin dilidir.

Robot bir nesneye yaklaşırken yalnızca mesafeyi değil, açıyı da optimize eder. Çünkü yanlış açı, doğru mesafeyi bile işe yaramaz hale getirebilir. Bu nedenle trigonometri, robotik sistemlerde yönelimin matematiksel karşılığıdır.

Ayrıca çoklu eklemli sistemlerde (örneğin robot kolları) her eklem bir dönüşüm yaratır. Bu dönüşümler zincirlenerek son noktanın konumunu belirler. Bu süreç “kinematik zincir” olarak bilinir.

Trigonometri olmadan bu zincir çözülemez.

Türev: Değişimin Matematiği

Türev, değişimin matematiksel ifadesidir. Bir büyüklüğün zamana veya başka bir değişkene göre nasıl değiştiğini ölçer. Robotik ve yapay zekâ açısından türev, hareketin ve öğrenmenin kalbidir.

Bir robotun konumu zamanla değişir. Bu değişimin hızı türevdir.

$$v(t) = \frac{dx(t)}{dt}$$

Bu ifade, hızın konumun değişim oranı olduğunu söyler. Robotik sistemlerde bu yalnızca fiziksel bir gerçek değil, aynı zamanda kontrol stratejisinin temelidir.

Bir robot kolu hareket ederken yalnızca nerede olduğunu değil, ne kadar hızlı değiştiğini de hesaba katar. Çünkü kontrol sistemleri değişimi yönetir.

Türev aynı zamanda öğrenmenin temelidir. Sinir ağlarında hata fonksiyonunun minimize edilmesi, türev hesapları üzerinden yapılır. Gradient descent adı verilen yöntem, hatanın en hızlı azaldığı yönü bulmak için türevleri kullanır.

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Bu ifade, öğrenmenin matematiksel özüdür. Sistem parametrelerini, hatayı azaltacak yönde günceller.

Buradaki sezgi şudur: öğrenme, değişimi takip etmektir.

Türev olmadan robotik sistemler durağan kalırdı. Çünkü değişim anlaşılmazdı.

Üç Kavramın Birleşimi: Robotik Düşünmenin Temeli

Vektör, trigonometri ve türev ayrı ayrı kavramlar gibi görünse de robotik sistemlerde tek bir bütün oluşturur. Vektörler uzayı temsil eder, trigonometri yönelimi tanımlar, türev ise değişimi açıklar.

Bir robot kolu hareket ederken aslında şu üç süreci aynı anda yürütür:

- Nerede olduğunu (vektör)
- Hangi yöne baktığını (trigonometri)
- Nasıl değiştiğini (türev)

Bu üçlü birleşim olmadan robotik davranış mümkün değildir.

Öğrenen sistemlerde bu yapı daha da önem kazanır. Çünkü sistem yalnızca statik bir durumu değil, dinamik bir dünyayı modellemek zorundadır.

Robotik düşünme, bu nedenle matematiksel bir dilden çok daha fazlasıdır; dünyayı temsil etmenin bir yoludur.

Sonuç

Matematiksel sezgiler, robotik ve yapay zekânın görünmeyen omurgasını oluşturur. Vektörler dünyayı, trigonometrik ilişkiler yönü, türev ise değişimi temsil eder.

Bu üç kavram birlikte düşünüldüğünde, robotik sistemlerin neden çalıştığı daha net anlaşılır: çünkü dünya zaten bu üç yapının etkileşimiyle işler.

Ve belki de en önemli fikir şudur: Matematik, robotlar için bir hesaplama aracı değil; hareketin, algının ve öğrenmenin doğrudan kendisidir.

EK B — GitHub ile Proje Yönetimi

Yapay zekâ, robotik ve yazılım tabanlı araştırma projeleri artık tek bir kişinin zihninde değil, dağıtık ekiplerin ortak üretim alanında şekillenmektedir. Bu nedenle modern araştırma pratiğinde yalnızca algoritmalar değil, aynı zamanda proje yönetim araçları da belirleyici hale gelmiştir. Bu bağlamda GitHub, yalnızca bir kod depolama platformu değil, aynı zamanda bilgi üretiminin organize edildiği bir işbirliği ekosistemidir.

GitHub ile proje yönetimi, özellikle yapay zekâ ve robotik projelerinde üç temel ihtiyacı karşılar: sürüm kontrolü, işbirliği ve izlenebilirlik. Bu üç unsur, karmaşık sistemlerin sürdürülebilir şekilde geliştirilmesi için kritik öneme sahiptir.

Sürüm Kontrolü: Zaman İçinde Kodun Hafızası

Yapay zekâ projeleri sürekli değişir. Modeller güncellenir, veri setleri genişler, hiperparametreler ayarlanır ve mimariler yeniden tasarlanır. Bu sürekli değişim içinde, geçmişe dönmek çoğu zaman hayati önem taşır.

Sürüm kontrol sistemleri, bu değişimi kayıt altına alır. Her değişiklik bir "commit" olarak saklanır ve sistemin geçmişi adım adım izlenebilir hale gelir.

Bu yapı, yalnızca teknik bir özellik değildir; aynı zamanda bilimsel bir hafıza sistemidir. Bir modelin neden daha iyi ya da daha kötü performans verdiğini anlamak için geçmiş değişikliklerin izlenmesi gerekir.

GitHub bu süreci standartlaştırır ve ekiplerin ortak bir zaman çizelgesi üzerinde çalışmasını sağlar.

İşbirliği: Dağıtık Zekânın Organizasyonu

Modern yapay zekâ projeleri tek bir araştırmacı tarafından geliştirilemeyecek kadar karmaşıktır. Veri mühendisleri, makine öğrenmesi uzmanları, robotik mühendisler ve yazılım geliştiriciler aynı sistem üzerinde çalışır.

GitHub, bu dağıtık yapıyı organize eder.

Branch yapıları sayesinde farklı ekipler aynı proje üzerinde paralel olarak çalışabilir. Pull request mekanizması, değişikliklerin kontrol edilmesini ve tartışılmasını sağlar.

Bu süreç, yalnızca teknik bir akış değildir; aynı zamanda kolektif düşünmenin bir biçimidir. Her kod değişikliği, ekip içinde bir fikir önerisi anlamına gelir.

Kod inceleme (code review) süreci, hataları azaltırken aynı zamanda bilgi paylaşımını artırır. Bu sayede sistem yalnızca yazılım olarak değil, aynı zamanda öğrenen bir organizasyon olarak gelişir.

İzlenebilirlik: Bilimsel Şeffaflık

Yapay zekâ projelerinde en kritik problemlerden biri tekrar üretilebilirliktir. Bir modelin nasıl eğitildiği, hangi veriyle çalıştığı ve hangi parametrelerle optimize edildiği net değilse, sonuçlar bilimsel olarak anlamını kaybeder.

GitHub, bu problemi çözmek için güçlü bir izlenebilirlik yapısı sunar.

Kod, veri işleme adımları ve deney sonuçları aynı ortamda saklanabilir. Bu sayede bir araştırma yalnızca sonuçlarıyla değil, süreciyle de değerlendirilebilir hale gelir.

Bu yaklaşım, yapay zekâ araştırmalarını daha bilimsel ve şeffaf bir yapıya taşır.

Proje Yapısı: Robotik ve AI Sistemlerinde Düzen

Büyük ölçekli robotik projelerde kod yapısı en az algoritma kadar önemlidir. Karmaşık sistemler modüler hale getirilmezse sürdürülebilir olmaz.

GitHub üzerinde tipik bir yapay zekâ projesi şu bileşenlerden oluşur:

- veri işleme modülleri
- model mimarileri
- eğitim scriptleri
- değerlendirme araçları
- simülasyon ortamları

Bu yapı, sistemin hem geliştirilebilir hem de test edilebilir olmasını sağlar.

Özellikle robotik projelerde simülasyon ve gerçek dünya kodları çoğu zaman ayrı tutulur. Bu ayrım, Sim-to-Real geçişinin daha kontrollü yapılmasını sağlar.

Deney Yönetimi: Bilimsel Çalışmanın Dijital Hali

Yapay zekâ araştırmalarında deneyler merkezi bir rol oynar. Farklı modeller, veri setleri ve hiperparametreler sürekli test edilir.

GitHub bu deneylerin yönetimini kolaylaştırır.

Her deney bir commit veya branch olarak temsil edilebilir. Böylece hangi değişikliğin hangi sonucu ürettiği net şekilde görülebilir.

Bu yapı, geleneksel laboratuvar defterinin dijital versiyonudur.

CI/CD: Otomatik Test ve Sürekli Gelişim

Modern GitHub projelerinde sürekli entegrasyon (CI) ve sürekli dağıtım (CD) sistemleri önemli bir yer tutar. Her kod değişikliği otomatik olarak test edilir ve doğrulanır.

Bu süreç, özellikle yapay zekâ sistemlerinde kritik öneme sahiptir. Çünkü küçük bir kod değişikliği bile model performansını ciddi şekilde etkileyebilir.

Otomatik testler sayesinde hatalar erken aşamada tespit edilir.

Açık Kaynak Kültürü: Kolektif Zekâ

GitHub aynı zamanda açık kaynak kültürünün merkezidir. Yapay zekâ ve robotik alanındaki birçok önemli proje açık kaynak olarak geliştirilir.

Bu durum, bilgi paylaşımını hızlandırır ve inovasyonu artırır.

Açık kaynak sistemler, bireysel zekânın ötesinde kolektif bir zekâ üretir. Binlerce geliştirici aynı proje üzerinde çalışarak sürekli gelişen bir ekosistem oluşturur.

Robotik Projelerde GitHub Kullanımı

Robotik projelerde GitHub yalnızca kod değil, aynı zamanda sistem mimarisinin tamamını içerir. Sensör verileri, kontrol algoritmaları, simülasyon modelleri ve donanım entegrasyonları aynı platformda organize edilir.

Bu yapı, özellikle büyük robotik sistemlerde hata ayıklamayı ve geliştirmeyi kolaylaştırır.

Örneğin bir robot kolu projesinde:

- kontrol algoritmaları bir repo'da
- simülasyon ortamı başka bir repo'da
- veri toplama scriptleri ayrı bir modülde

tutulabilir.

Bu modüler yapı, karmaşık sistemlerin yönetimini mümkün kılar.

Sonuç

GitHub ile proje yönetimi, modern yapay zekâ ve robotik geliştirme süreçlerinin temel omurgalarından biridir. Sadece kod saklama aracı değil, aynı zamanda kolektif zekânın organize edildiği bir platformdur.

Sürüm kontrolü, işbirliği, izlenebilirlik ve otomasyon bir araya geldiğinde, yazılım geliştirme bireysel bir çabadan çıkar ve bilimsel bir üretim sürecine dönüşür.

Ve belki de en önemli fikir şudur: Modern yapay zekâ projeleri yalnızca algoritmalarla değil, bu algoritmaların nasıl organize edildiğiyle de tanımlanır.

EK C — Öğretmenler İçin Kılavuz

Yapay zekâ, robotik ve veri temelli sistemler artık yalnızca mühendislerin ve araştırmacıların alanı olmaktan çıkmış, eğitim dünyasının merkezine yerleşmiştir. Bu dönüşüm, öğretmenin rolünü ortadan kaldırmaz; tam tersine onu yeniden tanımlar. Öğretmen artık yalnızca bilgi aktaran kişi değil, öğrenme sürecini tasarlayan, yöneten ve yönlendiren bir "bilişsel mimar" haline gelir.

Bu ek bölüm, özellikle matematik, bilgisayar bilimi, mühendislik ve fen bilimleri alanlarında ders veren öğretmenler için hazırlanmıştır. Amaç, yapay zekâ ve robotik konularının nasıl öğretileceğine dair bir çerçeve sunmak ve öğrencilerin bu alanlarda yalnızca teorik bilgi değil, sezgisel ve uygulamalı anlayış geliştirmesini sağlamaktır.

1. Öğretmenin Değişen Rolü

Geleneksel eğitim modelinde öğretmen, bilgiyi aktaran merkez konumundaydı. Öğrenci ise bu bilginin pasif alıcısıydı. Ancak yapay zekâ çağında bilgi artık sabit değildir; sürekli güncellenir, genişler ve dönüşür. Bu nedenle öğretmenin rolü de değişmiştir.

Artık öğretmen:

- Bilgiyi aktaran değil, bilgiye erişimi öğreten
- Cevapları veren değil, soruları tasarlayan
- Ezberlettiren değil, düşündüren kişidir

Bu değişim, eğitimde temel bir paradigma dönüşümüdür.

2. Sezgi Temelli Öğretim

Yapay zekâ ve robotik konularında en büyük hata, öğrencileri doğrudan matematiksel formalizme boğmaktır. Oysa bu alanların özü sezgidir.

Örneğin bir vektör anlatılırken yalnızca formül verilmemeli, yön ve büyüklük kavramı fiziksel örneklerle ilişkilendirilmelidir. Bir robot kolunun hareketi anlatılırken yalnızca kinematik denklemler değil, gerçek dünyadaki hareket hissi de aktarılmalıdır.

Sezgi, matematiğin ön koşuludur.

Öğrenci önce “neden”i anlamalı, sonra “nasıl”ı öğrenmelidir.

3. Deneyim Tabanlı Öğrenme

Yapay zekâ ve robotik, doğası gereği deneyimsel alanlardır. Bu nedenle öğretim süreci de deneyime dayanmalıdır.

Öğrenciler yalnızca teorik olarak algoritmaları öğrenmemeli, aynı zamanda onları uygulamalıdır. Küçük simülasyonlar, kodlama projeleri ve robotik deney setleri bu sürecin temel araçlarıdır.

Örneğin pekiştirmeli öğrenme anlatılırken öğrenciler basit bir ortamda bir ajanın ödül toplamasını gözlemlemelidir. Hata yapmanın öğrenme sürecinin doğal bir parçası olduğu bizzat görülmelidir.

Bu yaklaşım, öğrenmeyi soyut bir süreç olmaktan çıkarır.

4. Hata Kültürü

Eğitimde en kritik dönüşümlerden biri “hata” kavramına bakıştır. Geleneksel sistemlerde hata olumsuz bir sonuç olarak görülür. Oysa yapay zekâda hata, öğrenmenin temel kaynağıdır.

Bir öğrencinin yanlış yapması, aslında sistemin çalıştığını gösterir.

Öğretmenler hata kültürünü teşvik etmelidir:

- Yanlış cevap cezalandırılmamalı
- Hatalar analiz edilmelidir
- Her hata bir öğrenme fırsatı olarak görülmelidir

Bu yaklaşım, öğrencinin deney yapma cesaretini artırır.

5. Proje Tabanlı Öğrenme

Yapay zekâ ve robotik konularında en etkili öğretim yöntemi proje tabanlı öğrenmedir. Öğrenciler küçük ama gerçek problemler üzerinde çalışmalıdır.

Örneğin:

- Basit bir robot kol simülasyonu
- Görüntü sınıflandırma modeli
- Oyun oynayan bir ajan
- Sensör verisi analizi

Bu tür projeler, teoriyi pratiğe dönüştürür.

Öğrenci yalnızca “öğrenmez”, aynı zamanda “üretir”.

6. Disiplinlerarası Yaklaşım

Yapay zekâ tek bir disiplin değildir. Matematik, bilgisayar bilimi, fizik, istatistik ve hatta felsefe ile iç içedir.

Bu nedenle öğretmenler derslerini disiplinlerarası bir bakış açısıyla tasarlamalıdır.

Örneğin bir robotik dersi yalnızca mühendislik değil, aynı zamanda etik soruları da içermelidir:

- Bir robotun kararından kim sorumludur?
- Otonomi ne kadar olmalıdır?
- İnsan–makine ilişkisi nasıl olmalıdır?

Bu sorular öğrencinin düşünme kapasitesini genişletir.

7. Görselleştirme ve Simülasyon

Yapay zekâ kavramları çoğu zaman soyuttur. Bu nedenle görselleştirme araçları kritik öneme sahiptir.

Öğrenciler:

- vektör alanlarını
- sinir ağlarını
- öğrenme süreçlerini
- robot hareketlerini

görsel olarak incelemelidir.

Simülasyon ortamları, öğrenmeyi somutlaştırır ve sezgiyi güçlendirir.

8. Öğrenmeyi Öğretmek

Modern eğitimde en önemli hedeflerden biri “öğrenmeyi öğretmek”tir. Çünkü bilgi hızla değişmektedir.

Öğrenciler:

- nasıl araştırma yapılacağını
- nasıl model kurulacağını
- nasıl hata analiz edileceğini

öğrenmelidir.

Bu beceriler, spesifik bir konudan daha değerlidir.

9. Yapay Zekâ Araçlarının Sınıfta Kullanımı

Günümüzde yapay zekâ araçları eğitimde aktif olarak kullanılabilir. Kod üretim araçları, simülasyon sistemleri ve veri analiz platformları öğretimi destekler.

Ancak bu araçlar öğrencinin yerine geçmemelidir. Amaç, öğrencinin düşünme sürecini desteklemektir.

Öğrenci önce düşünmeli, sonra araçtan yardım almalıdır.

10. Geleceğe Hazırlık

Eğitim yalnızca bugünün ihtiyaçlarını değil, geleceğin belirsizliklerini de kapsamalıdır.

Öğrenciler:

- değişen teknolojilere uyum sağlayabilmeli
- yeni sistemleri öğrenebilmeli

- belirsizlikle çalışabilmelidir

Bu nedenle müfredat sabit değil, esnek olmalıdır.

Sonuç

Öğretmenler için yapay zekâ ve robotik kılavuzu, yalnızca teknik bir rehber değil, aynı zamanda bir eğitim felsefesidir. Bu alanlarda başarılı öğretim, bilgi aktarımından çok düşünme biçimi kazandırmaya dayanır.

Öğrenciye öğretilmesi gereken şey bir algoritma değil, bir bakış açısıdır.

Ve belki de en önemli fikir şudur: Geleceğin eğitimi, bilgiyi ezberleten değil, öğrenmeyi sürekli hale getiren

EK D — Kulüpler İçin 1 Yıllık Uygulama Planı

Yapay zekâ ve robotik kulüpleri, bir okulun veya üniversitenin yalnızca "ek etkinliği" değildir; doğru kurgulandığında, öğrencinin düşünme biçimini dönüştüren küçük bir araştırma laboratuvarına dönüşür. Bu tür kulüplerin en büyük problemi çoğu zaman motivasyon eksikliği değil, yönsüzlüktür. Öğrenciler heyecanlı başlar ama proje sürekliliği sağlanamadığında öğrenme dağılır.

Bu nedenle burada sunulan 1 yıllık plan, rastgele etkinlikler listesi değil, aşamalı olarak derinleşen bir öğrenme mimarisidir. Amaç, öğrenciyi basit simülasyonlardan başlayarak gerçek dünya robotik problemlerine taşıyan bir gelişim hattı oluşturmaktır.

Plan dört çeyreğe ayrılmıştır. Her çeyrek, farklı bir bilişsel ve teknik odak içerir.

1. Çeyrek (Ay 1–3): Temel Sezgi ve Dijital Düşünme

Bu dönem, öğrencilerin yapay zekâ ve robotik dünyasına giriş yaptığı aşamadır. Amaç teknik uzmanlık kazandırmak değil, sezgisel temel oluşturmaktır.

Öğrenciler bu dönemde şu konularla tanıştırılır:

- Vektör mantığı ve uzay sezgisi
- Temel Python programlama
- Basit simülasyon ortamları
- Görselleştirme araçları

Bu aşamada en önemli hedef, öğrencinin "algoritma" kavramını korkulacak bir şey olmaktan çıkarmaktır. Kod yazmak bir mühendislik değil, bir düşünme biçimi olarak sunulmalıdır.

Örnek uygulamalar:

- 2D hareket eden bir ajan simülasyonu

- Basit labirent çözme problemi
- Rastgele hareket eden robot davranışları

Bu projelerde başarıdan çok keşif önemlidir.

2. Çeyrek (Ay 4–6): Öğrenme Sistemlerine Giriş

İkinci çeyrek, öğrencilerin yapay zekânın temel kavramı olan öğrenme fikriyle tanıştığı dönemdir.

Burada odak noktası pekiştirmeli öğrenme ve temel makine öğrenmesi kavramlarıdır.

Öğrenciler şunları öğrenir:

- Ödül ve ceza mekanizması
- Politika kavramı
- Basit sinir ağları
- Deneme-yanılma süreçleri

Bu dönemde en önemli kazanım, sistemlerin sabit değil öğrenen yapılar olduğunu anlamaktır.

Örnek projeler:

- Bir ajanın hedefe ulaşmayı öğrenmesi
- Basit oyun oynayan yapay zekâ (örneğin grid world)
- Q-learning ile karar verme simülasyonu

Bu aşamada öğrenciler ilk kez “hata = öğrenme” ilişkisini deneyimsel olarak görür.

3. Çeyrek (Ay 7–9): Robotik Simülasyon ve Fiziksel Sezgi

Üçüncü çeyrek, öğrencilerin soyut sistemlerden fiziksel dünyaya geçtiği aşamadır.

Burada amaç, robotik düşünmenin temelini oluşturan fiziksel sezgiyi geliştirmektir.

Öğrenciler şu konularla çalışır:

- Simülasyon ortamları
- Temel robot kinematiği
- Nesne kavrama ve hareket
- Sensör verisi yorumlama

Bu aşamada özellikle fizik motorları ve sim-to-real mantığı sezgisel olarak öğretilmelidir.

Örnek projeler:

- Robot kol simülasyonu ile nesne taşıma
- Çarpışma önleme davranışı
- Görsel algıya dayalı yön bulma

Bu çeyrekte öğrenciler ilk kez “dijital kararın fiziksel karşılığı” olduğunu görür.

4. Çeyrek (Ay 10–12): Otonomi ve Grup Projeleri

Son çeyrek, öğrencilerin bireysel öğrenmeden kolektif üretime geçtiği aşamadır. Burada amaç, küçük araştırma projeleri geliştirmektir.

Bu dönemde öğrenciler:

- Çoklu ajan sistemleri
- Otonom karar verme
- Basit robotik görevler
- Takım tabanlı proje geliştirme

konularında çalışır.

Örnek final projeleri:

- Birden fazla ajanın ortak hedefe ulaşması
- Otonom araç simülasyonu
- Nesne tanıyan ve sınıflandıran robot sistemi
- Basit akıllı şehir simülasyonu

Bu aşamada öğrenciler yalnızca teknik bilgi değil, proje yönetimi ve işbirliği becerisi de kazanır.

Kulüp Yapısı ve Organizasyon

Başarılı bir yapay zekâ kulübü yalnızca plan değil, iyi bir organizasyon yapısı gerektirir.

Önerilen yapı:

- Takım liderleri (proje koordinasyonu)
- Teknik ekip (kod ve algoritma)
- Veri ekibi (veri hazırlama ve analiz)
- Simülasyon ekibi (ortam tasarımı)

Bu yapı, öğrencilerin gerçek bir mühendislik ekibi deneyimi yaşamasını sağlar.

Haftalık Çalışma Modeli

Kulüpler için önerilen haftalık düzen:

- 1 gün: Teorik ders ve tartışma
- 1 gün: Kodlama atölyesi
- 1 gün: Proje geliştirme
- 1 gün: Sunum ve değerlendirme

Bu döngü, öğrenmenin sürekli olmasını sağlar.

Değerlendirme Sistemi

Klasik sınav sistemi yerine proje tabanlı değerlendirme önerilir.

Öğrenciler şu kriterlere göre değerlendirilir:

- Problem çözme yaklaşımı
- Kod kalitesi
- Takım çalışması
- Yaratıcılık
- Süreklilik

Bu yaklaşım, rekabetten çok gelişimi teşvik eder.

Mentorluk Sistemi

Kulüp yapısında mentorlar kritik rol oynar. Mentorlar:

- Yol gösterir
- Sorun çözmez, yönlendirir
- Soru sormayı teşvik eder

Bu yaklaşım, öğrencinin bağımsız düşünmesini sağlar.

Sonuç

Yapay zekâ ve robotik kulüpleri, doğru yapılandırıldığında yalnızca teknik eğitim değil, aynı zamanda düşünme kültürü kazandıran ortamlara dönüşür. Bu 1 yıllık plan, öğrenciyi sıfırdan alıp karmaşık sistemleri anlayabilen bir noktaya taşımayı hedefler.

Ve belki de en önemli fikir şudur: Bir kulübün başarısı, öğrettiği bilgi miktarıyla değil, öğrencide yarattığı düşünme dönüşümüyle ölçülür.

KAYNAKÇA

Aşağıdaki kaynakça, bu kitapta ele alınan yapay zekâ, robotik, pekiştirmeli öğrenme, çoklu ajan sistemleri ve simülasyon temelli öğrenme konularının kuramsal ve pratik arka planını oluşturan temel eserlerden ve güncel araştırma alanlarından derlenmiştir.

Temel Yapay Zekâ ve Makine Öğrenmesi

- Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Pearson.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Murphy, K. P. (2022). *Probabilistic Machine Learning: An Introduction*. MIT Press.

Pekiştirmeli Öğrenme ve Karar Verme

- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research*.
- Silver, D. et al. (2016). "Mastering the game of Go with deep neural networks and tree search". *Nature*.

Robotik ve Fiziksel Zekâ

- Siciliano, B., & Khatib, O. (Eds.). (2016). *Springer Handbook of Robotics*. Springer.
- Craig, J. J. (2005). *Introduction to Robotics: Mechanics and Control*. Pearson.
- Lynch, K. M., & Park, F. C. (2017). *Modern Robotics*. Cambridge University Press.
- Hauser, K. (2014). "Learning and Planning in Robotics". *Annual Review of Control, Robotics, and Autonomous Systems*.

Simülasyon, Fizik Motorları ve Sim-to-Real

- Tobin, J. et al. (2017). "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". *IROS*.

- Peng, X. B. et al. (2018). “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”. *ICRA*.
- OpenAI (2020). “Learning Dexterous In-Hand Manipulation”. *arXiv preprint*.
- NVIDIA Isaac Lab Documentation ve araştırma notları. NVIDIA Isaac Lab

Çoklu Ajan Sistemleri ve Oyun Teorisi

- Shoham, Y., & Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.
- Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley.
- Sutton, R. S. et al. (2000). “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. *NeurIPS*.

Matematiksel Temeller

- Strang, G. (2016). *Introduction to Linear Algebra*. Wellesley-Cambridge Press.
- Stewart, J. (2015). *Calculus: Early Transcendentals*. Cengage Learning.
- Lay, D. C. (2012). *Linear Algebra and Its Applications*. Pearson.

Yapay Zekâ Felsefesi ve Bilişsel Bilim

- Turing, A. M. (1950). “Computing Machinery and Intelligence”. *Mind*.
- Searle, J. (1980). “Minds, Brains, and Programs”. *Behavioral and Brain Sciences*.
- Dennett, D. C. (1991). *Consciousness Explained*. Little, Brown and Company.
- Floridi, L. (2019). *The Logic of Information*. Oxford University Press.

Açık Kaynak ve Yazılım Ekosistemi

- GitHub Documentation and Engineering Practices. GitHub
- Newman, S. (2015). *Building Microservices*. O’Reilly Media.
- Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley.

Robotik Simülasyon ve Araçlar

- NVIDIA Isaac Sim / Isaac Lab teknik dokümantasyonu. Isaac Sim
- Coumans, E. et al. *PyBullet Physics Simulation*. PyBullet
- OpenAI Gym: Brockman, G. et al. (2016). “OpenAI Gym”. OpenAI Gym

Son Not

Bu kaynakça, kitapta ele alınan konuların doğrudan alıntılandığı bir liste değil, tematik olarak beslendiği bilimsel ve teknik literatürün bir haritasıdır. Kitapta sunulan bölümler, bu alanların kesişiminde oluşan modern araştırma yaklaşımının pedagojik bir yeniden yorumudur.